第五章 语法分析——自下而上分析

授课人：高珍

gaozhen@tongji.edu.cn

概述

- 自下而上分析法就是从输入串开始，逐步进行“归约”，直至归约到文法的开始符号。
- 从语法树的末端，步步向上“归约”，直到根结。

自上而下分析法:

开始符号S $\overset{*}{\Rightarrow}$ 输入串α （推导）

自下而上分析法:

输入串α $\overset{*}{\Rightarrow}$ 开始符号S （归约）

内容线索

- **自下而上分析基本问题**
- **规范规约**
- **算符优先分析方法**
- **LR分析方法**

归约

- 移进-归约法
  - 使用一个符号栈，把输入符号逐一移进栈，当栈顶形成某个产生式右部时，则将栈顶的这一部分替换（归约）为该产生式的左部符号。

例. 给定文法 G：

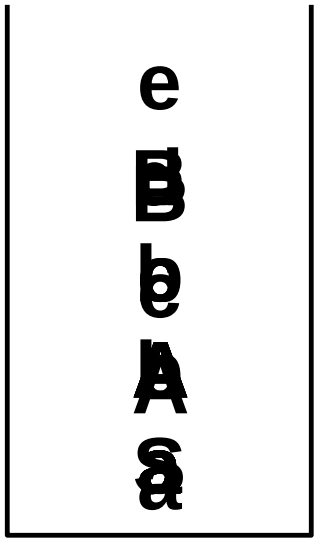
(1) S→aAcBe

(2) A→b

(3) A→Ab

(4) B→d　　输入串 abbcde是否为句子?

归约过程如下：



例. 给定文法 G：

(1) S→aAcBe

(2) A→b

(3) A→Ab

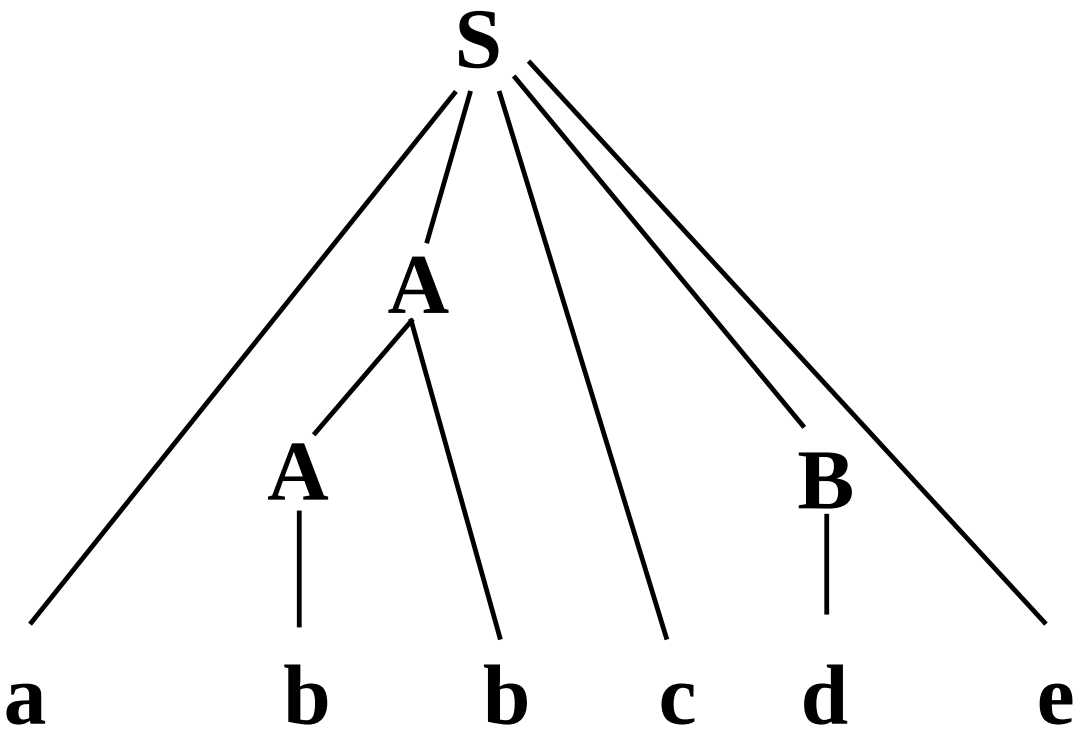
(4) B→d　　输入串 abbcde是否为句子?

归约过程如下：

步骤：	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
动作：	进	进	归	进	归	进	进	归	进	归
	a	b	(2)	b	(3)	c	d	(4)	e	(1)

									e
							d	B	B
				b		c	c	c	c
	b	A	A	A	A	A	A	A	A
a	a	a	a	a	a	a	a	a	a	S

分析树：用树表示“移进－归约”过程



自下而上分析的基本问题

- 如何找出或确定可规约串？
- 对找出的可规约串替换为哪一个非终结符号？

内容线索

- ✓ 自下而上分析基本问题
- **规范规约**
- 算符优先分析方法
- LR分析方法

短语

- 令G是一个文法，S是文法的开始符号，若αβδ是文法G的一个句型，如果有

$$S \overset{*}{\Rightarrow} \alpha A \delta \text{且 } A \overset{+}{\Rightarrow} \beta$$

则称β是句型αβδ相对于非终结符A的**短语**。

特别地，若 $A \Rightarrow \beta$，则称β是句型αβδ关于产生式 $A \rightarrow \beta$ 的**直接短语**。

- 一个句型的最左直接短语称为**句柄**。

例.设文法G（S）：　(1) S → aAcBe

(2) A → b

(3) A → Ab

(4) B → d

给出句型aAbcde的短语、直接短语、句柄。

由 S ⇒ aAcBe ⇒ aAcde ⇒ aAbcde

S ⇒ aAcBe ⇒ aAbcBe ⇒ aAbcde

短语：d, Ab, aAbcde

直接短语：d, Ab

句柄：Ab

句型语法树和句型的短语、直接短语、句柄

- 短语：句型语法树中每棵子树（某个结点连同它的所有子孙组成的树）的所有叶子结点从左到右排列起来形成一个相对于子树根的短语。
- 直接短语：只有父子两代的子树形成的短语。
- 句柄：语法树中最左那棵只有父子两代的子树形成的短语。

例.设文法G（S）：　(1) S → aAcBe

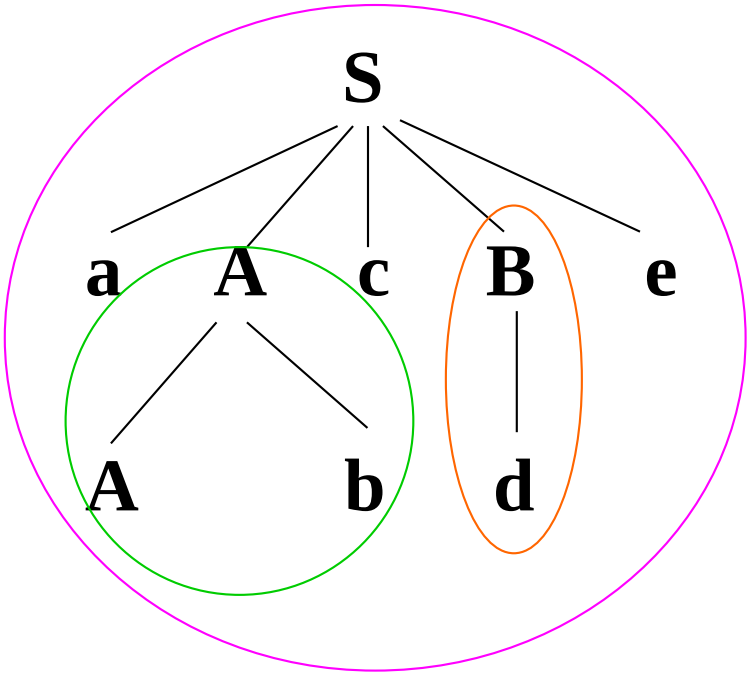
(2) A → b

(3) A → Ab

(4) B → d

给出句型aAbcde的短语、直接短语、句柄。

句型aAbcde的语法树为：



短语：d, Ab, aAbcde

直接短语：d, Ab

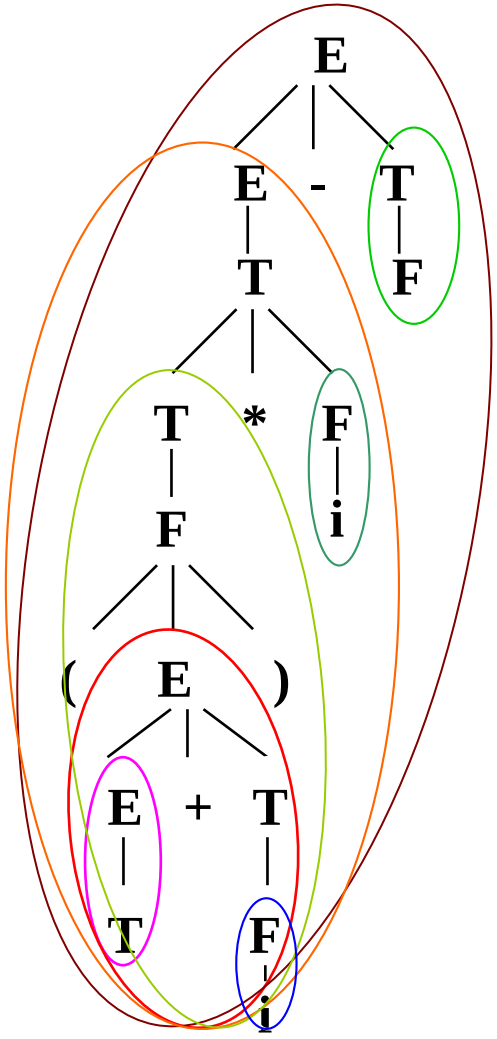
句柄：Ab

例. G（E）： E→E+T｜E-T｜T
T→T*F｜T/F｜F
F→ i ｜(E)

试找出句型 (T+ i) * i-F 的所有短语、直接短语和句柄

解：

短语	直接短语	句柄
(T+ i) * i -F	T	T
(T+ i) * i	i	
(T+ i)	i	
T+ i	F	
T		
i (左)		
i (右)		
F		



例. 给定文法G：E→E+E|E*E|(E)|i

给出句型E+E*E的句柄

解. (1) E ⇒ E+E ⇒ E+E*E

E*E是句柄

(2) E ⇒ E*E ⇒ E+E*E

E+E是句柄

注：二义性文法的句柄可能不唯一

规范归约

设α是文法G的一个句子，若序列$\alpha_n, \alpha_{n-1}, \cdots, \alpha_0$，满足：

（1）$\alpha_n = \alpha$；

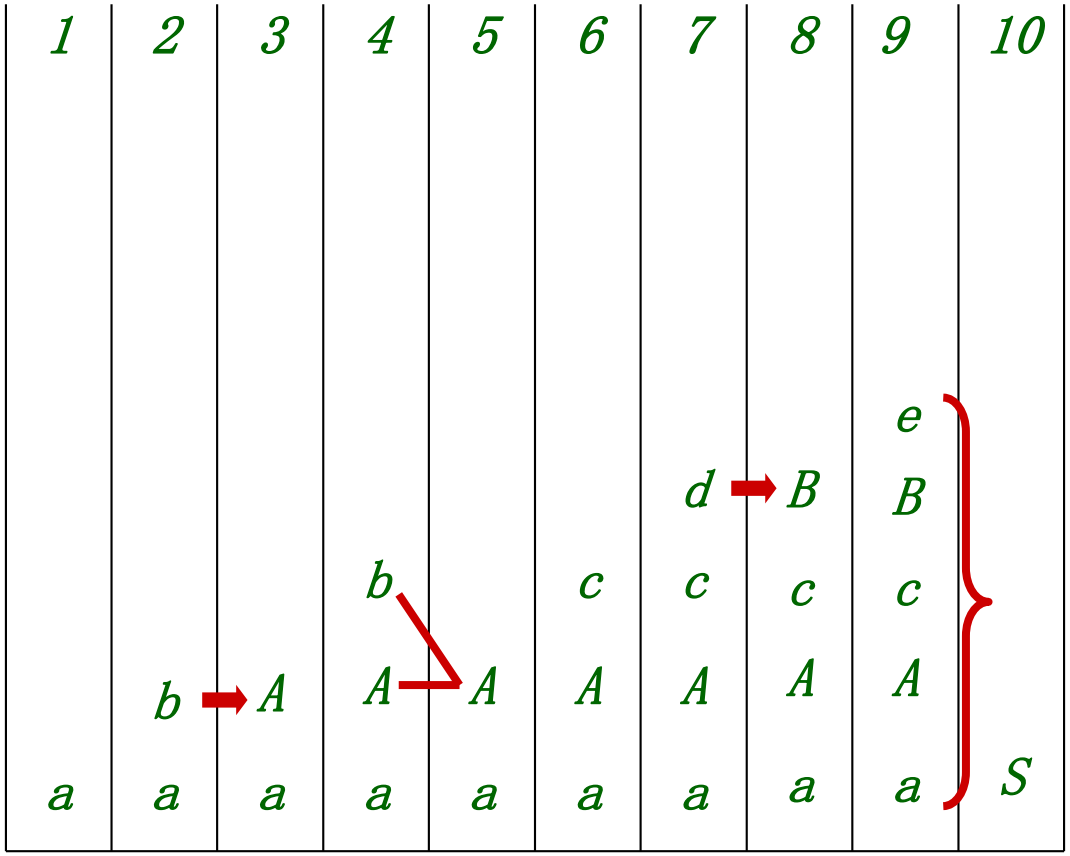
（2）$\alpha_0 = S$；

（3）对任意i，$0< i \leq n$，α_{i-1}是从α_i将**句柄**替换成相应产生左部符号而得到的

则称该序列是一个**规范归约**。

最右推导： S => aAcBe => aAcde =>aAbcde => a b b c d e

栈

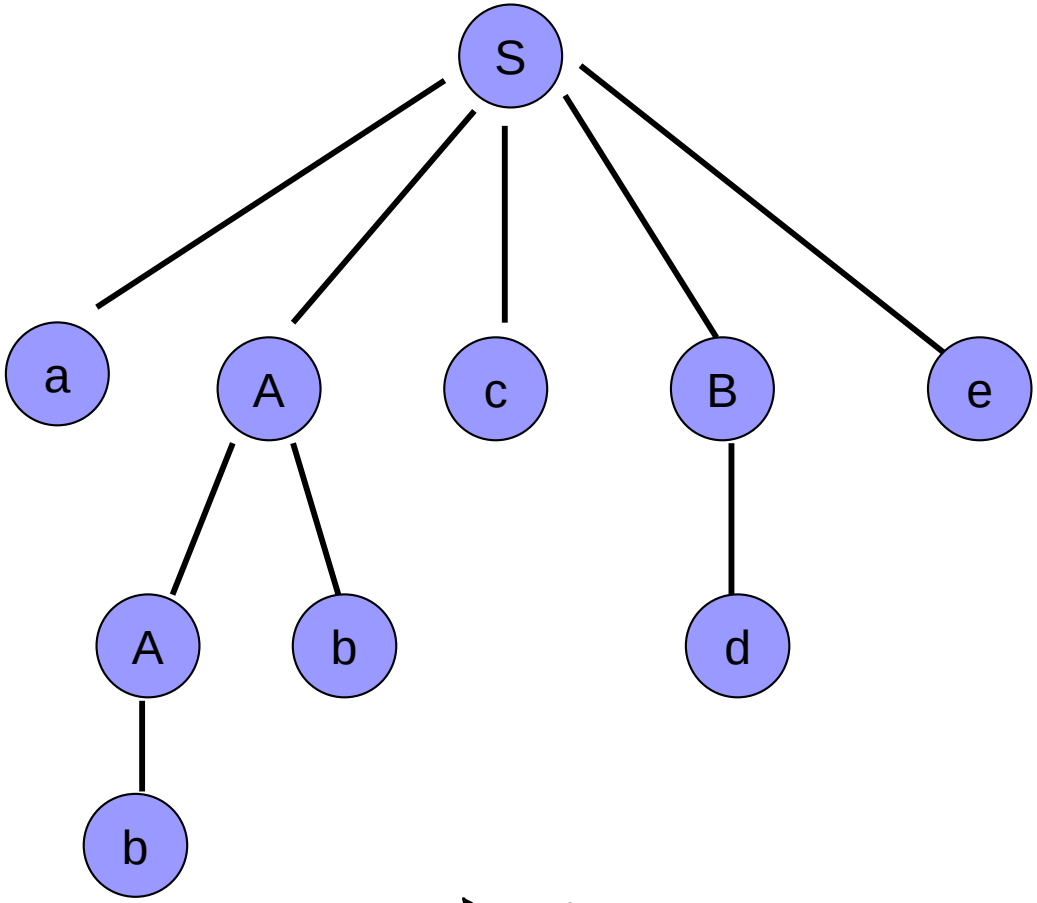


S → aAcBe　A → Ab

A → b　　B → d

输入串：abbcde

最左归约：a b b c d e => aAbcde => aAcde => aAcBe => S



分 析 树

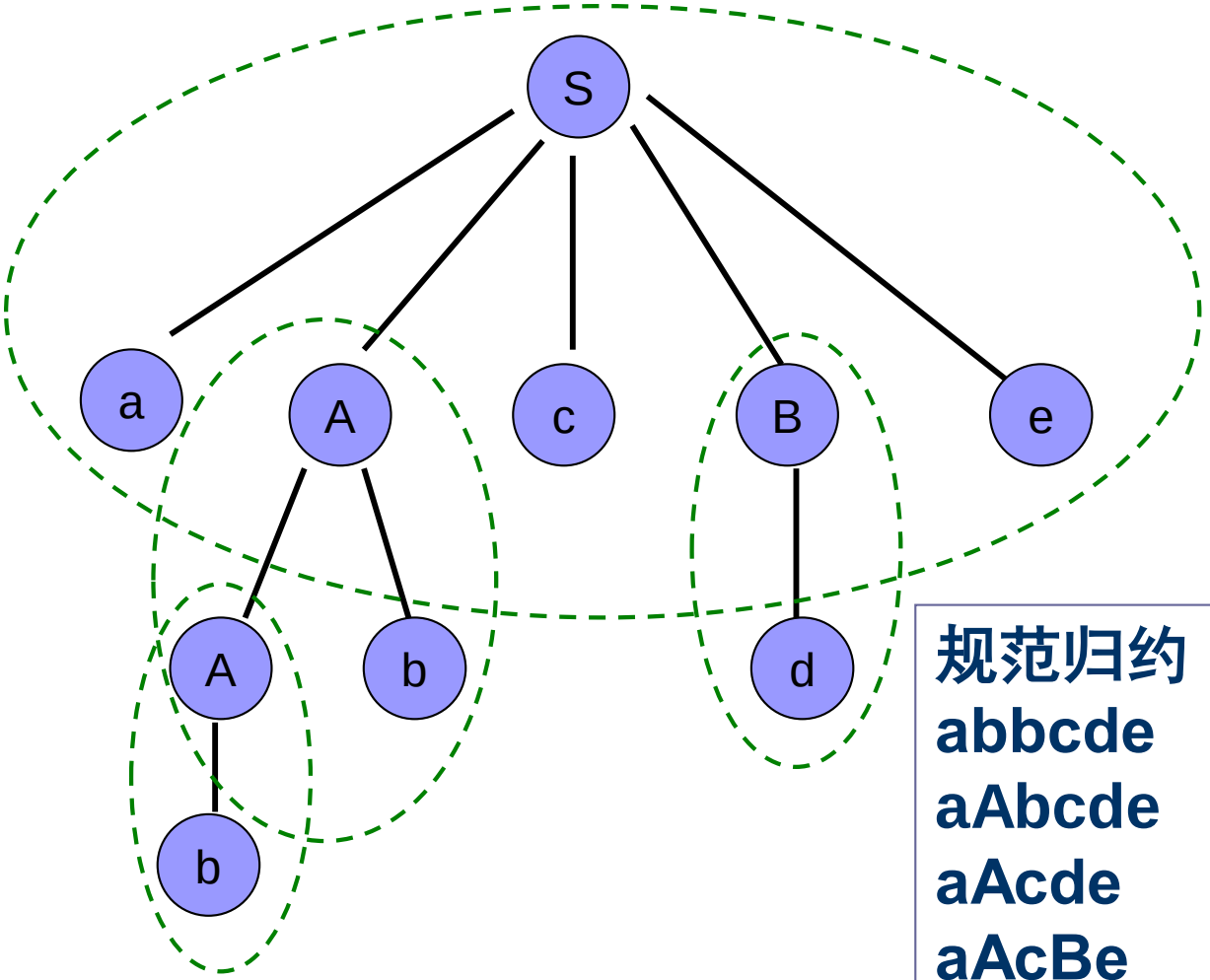
S->aAcBe A->Ab

A->b B->d

输入串：abbcde

句子abbcde 的规范归约过程如下：

——“剪枝”



文法G（S）

S→aAcBe

A→b

A→Ab

B→ d

规范归约	归约规则
abbcde	A→b
aAbcde	A→Ab
aAcde	B→d
aAcBe	S→aAcBe
S	

规范规约的基本问题

- 如何找出或确定可规约串——句柄？
- 对找出的可规约串——句柄替换为哪一个非终结符号？

符号栈的使用

- 实现移进-归约分析的一个方便途径是用一个**栈**和一个**输入缓冲区**，用#表示栈底和输入的结束

	栈	输入串
初始	#	w#

	栈	输入串
最终	#S	#

例. G: E→E+E | E*E | (E) | i 给出 $i_1*i_2+i_3$ 的移进归约过程

步骤	栈	输入串	动作
0	#	$i_1*i_2+i_3$#	预备
1	#i_1	$*i_2+i_3$#	移进
2	#E	$*i_2+i_3$#	归约E→i
3	#E*	i_2+i_3#	移进
4	#E*i_2	$+i_3$#	移进
5	#E*E	$+i_3$#	归约E→i
6	#E	$+i_3$#	归约E→E*E
7	#E+	i_3#	移进
8	#E+i_3	#	移进
9	#E+E	#	归约E→i
10	#E	#	归约E→E+E
11	#E	#	接受

随堂练习

- 求文法G1:E→E+E | E*E | (E) | i对于句子 $i_1*i_2+i_3$ 的另一个移进归约过程
- 对文法G2（如下），求句子i*i+i的规范规约步骤

  E→T|E+T

  T→F|T*F

  F→(E)|i

随堂练习

- **P133： 1题；2题**

语法分析的操作

- 移进
  - 下一输入符号移进栈顶,读头后移;
- 归约
  - 检查栈顶若干个符号能否进行归约，若能，就以产生式左部替代该符号串，同时输出产生式编号;
- 接收
  - 移进 - 归约的结局是栈内只剩下栈底符号和文法开始符号，读头也指向语句的结束符;
- 出错
  - 发现了一个语法错,调用出错处理程序

注：可归约的串在栈顶，不会在内部

内容线索

- ✓ 自下而上分析基本问题
- ✓ 规范规约
- ■ 算符优先分析方法
- ■ LR分析方法

算符优先分析方法

- 算符优先分析法是自下而上进行句型归约的一种分析方法。
- 定义**终结符**（算符）的优先关系，按终结符（算符）的优先关系控制自下而上语法分析过程（寻找“**可归约串**”和进行归约）。
- 不是规范归约，但分析速度快，适于表达式的语法分析。

优先关系

- 任何两个可能**相继出现**的终结符a和b（它们之间可能插有一个非终结符）的**优先关系**：
  - a ⋖ b　　a的优先级**低于**b
  - a ≐ b　　a的优先级**等于**b
  - a ⋗ b　　a的优先级**高于**b

注：这三种关系不同于数学中的<, =, >关系。

算符文法

- 一个文法，如果它的任一产生式右部都不含两个相继（并列）的非终结符，即不含如下形式的产生式右部：

  $$\ldots QR \ldots \quad , \quad Q, R \in V_N$$

  则称该文法为算符文法。

算符优先关系

- 设 G为算符文法且不含ε- 产生式， a, b∈ V_T, 算符间的优先关系定义为：
  - a ≐b 当且仅当G含有产生式 P →… ab… 或
    P →… aQb…
  - a ⋖ b 当且仅当G含有产生式 P →… aR … 且
    $R \overset{+}{\Rightarrow} b\ldots$ 或 $R \overset{+}{\Rightarrow} Qb\ldots$
  - a ⋗ b 当且仅当G含有产生式 P →… Rb… 且
    $R \overset{+}{\Rightarrow} \ldots a$ 或$R \overset{+}{\Rightarrow} \ldots aQ$

算符优先文法

- 如果一个算符文法G中的任何终结符对(a, b)**至多**满足下述关系之一

  $a \doteq b$,　　$a \lessdot b$,　　$a \gtrdot b$

  则称 G 为**算符优先文法**。

例. 给定文法G：E→E+E|E*E|(E)|i

其中：V_T={+, *, i, (,)} 。

G是算符文法

G是算符优先文法吗?

考察终结符对(+, *)

（1）因为E→E+E，且E⇒ E*E，所以

+ ⋖ *

（2）因为E→ E*E，且E⇒ E+E，所以

+ ⋗ *

G不是算符优先文法

例. 文法G: (1) E→E+T | T　　(2) T→T*F | F

(3) F→P↑F | P　　(4) P→(E) | i

算符优先关系为：

由(4): P→(E)　　∴ (≐)

由(1)(2): E→E+T, T ⇒ T*F　　∴ + ⋖ *

由(2) (3): T→T*F, F ⇒ P↑F　　∴ * ⋖ ↑

由(1): E→E+T, E ⇒ E+T　　∴ + ⋗ +

由(3): F→P↑F, F ⇒ P↑F　　∴ ↑ ⋖ ↑

由(4): P→(E), E ⇒ E+T　　∴ (⋖ +, + ⋗)

. . .

∴ G为算符优先文法

(# 看作终结符号，作为句子括号)

	+	*	↑	i	(	)	#
+	>	<	<	<	<	>	>
*	>	>	<	<	<	>	>
↑	>	>	<	<	<	>	>
i	>	>	>			>	>
(	<	<	<	<	<	=	
)	>	>	>			>	>
#	<	<	<	<	<		=

优先关系表的构造

- 通过检查G的每个产生式的每个候选式，可找出所有满足a ≐ b的终结符对。

a ≐b 当且仅当G含有产生式 P →… ab… 或
P →… aQb…

- 确定满足关系 ⋖ 和 ⋗ 的所有终结符对：

a ⋖ b 当且仅当G含有产生式 P →… aR … 且
$R \overset{+}{\Rightarrow} b\ldots$ 或 $R \overset{+}{\Rightarrow} Qb\ldots$

a ⋗ b 当且仅当G含有产生式 P →… Rb… 且
$R \overset{+}{\Rightarrow} \ldots a$ 或 $R \overset{+}{\Rightarrow} \ldots aQ$

FIRSTVT(P)和LASTVT(P)

设 $P \in V_N$，定义：

FIRSTVT (P) =

$\{ a \mid P \overset{+}{\Rightarrow} a \ldots$ 或 $P \overset{+}{\Rightarrow} Qa \ldots , a \in V_T , Q \in V_N \}$

LASTVT (P) =

$\{ a \mid P \overset{+}{\Rightarrow} \ldots a$ 或 $P \overset{+}{\Rightarrow} \ldots aQ , a \in V_T , Q \in V_N \}$

优先关系表的构造

- 有了这两个集合之后，就可以通过检查每个产生式的候选式确定满足关系 ⋖ 和 ⋗ 的所有终结符对。
  - 假定有个产生式的一个候选形为

    …aP…

    那么，对任何b∈FIRSTVT(P)，有 a ⋖ b。
  - 假定有个产生式的一个候选形为

    …Pb…

    那么，对任何a∈LASTVT(P)，有 a ⋗ b。

FIRSTVT(P)和LASTVT(P)构造

- **FIRSTVT (P) 构造**

  规则1: 若 P→a … 或 P→Qa … , 则a ∈FIRSTVT(P);

  规则2: 若a ∈FIRSTVT(Q) , 且P→Q … , 则

  a ∈FIRSTVT(P)。

- **LASTVT (P) 构造**

  规则1: 若 P→ … a 或 P→ … aQ , 则a ∈LASTVT(P);

  规则2: 若a ∈LASTVT(Q) , 且P→ … Q, 则

  a ∈LASTVT(P)。

FIRSTVT(P)的构造——数据结构

- **二维布尔矩阵F[P,a]和符号栈STACK**

$$\text{布尔矩阵}F[P,a] = \begin{cases} .T. & a \in FIRSTVT(P) \\ .F. & a \notin FIRSTVT(P) \end{cases}$$

栈 STACK: 存放使FIRSTVT 为真的符号对(P, a).

FIRSTVT(P)的构造——算法

- 把所有初值为真的数组元素F[P, a]的符号对(P, a)全都放在STACK之中。
- 如果栈STACK不空，就将栈顶逐出，记此项为(Q, a)。对于每个形如

$$P \rightarrow Q\ldots$$

的产生式，若F[P, a]为假，则变其值为真，且将(P, a)推进STACK栈。

- 上述过程必须一直重复，直至栈STACK拆空为止。

例. G: S→a | ^ | (T)
T→T,S | S

求 FIRSTVT(S), FIRSTVT(T)

M=	a	^	(	)	,
S	T	T	T	F	F
T	T	T	T	F	T

T ,
S (
S ^
S a

FIRSTVT(S) ={ a, ^, (}

FIRSTVT(T) ={ a, ^, (, ,}

类似地可得: LASTVT(S) ={a, ^,) }

LASTVT(T) ={a, ^,), , }

FIRSTVT主程序:

```
BEGIN
  FOR  每个非终结符P和终结符a  DO
          F[P,a] := FALSE;
  FOR  每个形如P→a… 或P→Qa … 的产生式  DO
          INSERT (P, a);
  WHILE  STACK非空  DO
  BEGIN
          把STACK 的顶项 (Q, a) 弹出;
          FOR  每条形如P→Q … 的产生式  DO
               INSERT (P, a);
   END OF WHILE;
END

PROCEDURE   INSERT(P,a);
     IF  NOT F[P,a]  THEN
     BEGIN
         F[P,a]:= true;
         把 (P, a)下推进STACK栈
      END;
```

构造优先关系表算法

FOR 每条产生式$P \rightarrow X_1X_2 \ldots X_n$ DO

  FOR i:=1 TO n-1 DO

  BEGIN

    IF X_i和X_{i+1}均为终结符 THEN 置$X_i \doteq X_{i+1}$

    IF $i \leq n-2$且X_i和X_{i+2}都为终结符

        但X_{i+1}为非终结符 THEN 置$X_i \doteq X_{i+2}$;

    IF X_i为终结符而X_{i+1}为非终结符 THEN

      FOR FIRSTVT(X_{i+1})中的每个a DO

        置 $X_i \lessdot a$;

    IF X_i为非终结符而X_{i+1}为终结符 THEN

      FOR LASTVT(X_i)中的每个a DO

        置 $a \gtrdot X_{i+1}$

END

例. G:　S→a | ^ | (T)　　T→T,S | S

FIRSTVT(S) = { a, ^, (}　FIRSTVT(T) = { a, ^, (, , }

LASTVT(S) = { a, ^,) }　LASTVT(T) = { a, ^,), , }

优先关系	a	^	(	)	,
a				▭	▭
^				▭	▭
(	▭	▭	▭	▭	▭
)				▭	▭
,	▭	▭	▭	▭	▭

最左素短语——算符优先分析中的可归约串

- 素短语
  - 是一个短语，它至少含有一个终结符且除它自身之外不含有任何更小的素短语。
- 最左素短语
  - 处于句型最左边的那个素短语。

例. 对文法G:

(1) E→E+T | T

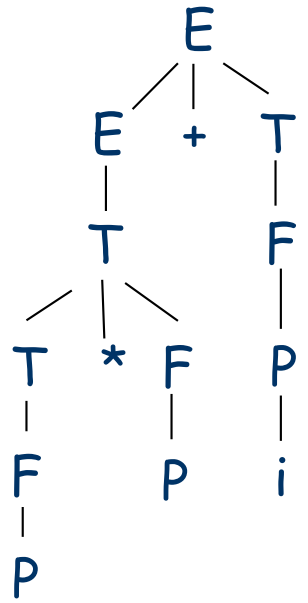
(2) T→T*F | F

(3) F→P↑F | P

(4) P→(E) | i

求句型P*P+i的最左素短语

解: 句型的语法树为:



句型的短语:P, P*P, i, P*P+i

素短语:P*P, i

最左素短语:P*P

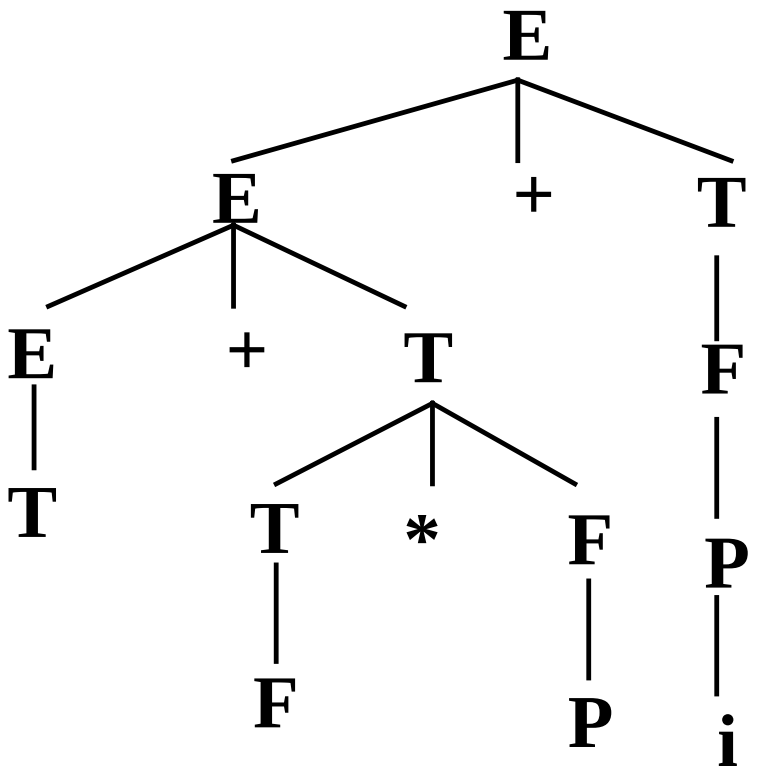
例. 对文法G:

(1) E→E+T | T

(2) T→T*F | F

(3) F→P↑F | P

(4) P→(E) | i



句型：T+F*P+i

短语： T，F，P，i，F*P，T+F*P，T+F*P+i

直接短语：T，F，P，i

句柄：T

素短语：F*P，i

最左素短语：F*P

算符优先文法的最左素短语

- **算符优先文法句型**(括在两个＃之间)的一般形式为:

  $\#N_1a_1N_2a_2 \ldots N_na_nN_{n+1}\#$

  其中:$a_i \in V_T$, $N_i \in V_N$ (可有可无)

- 一个算符优先文法G的任何句型的**最左素短语**是满足下列条件的最左子串$N_ja_j \ldots N_ia_i N_{i+1}$

  $a_{j-1} \lessdot a_j$

  $a_j \doteq a_{j+1} \doteq \ldots \doteq a_{i-1} \doteq a_i$

  $a_i \gtrdot a_{i+1}$

例. 句型#P*P+i#中，$\# \lessdot *$，$* \gtrdot +$，所以P*P是最左素短语

算符优先分析算法

1）将输入串依此逐个存入符号栈S中，直到符号栈顶元素S_k与下一个待输入的符号a有优先关系$S_k>a$为止；

2）至此，最左素短语尾符号S_k已在符号栈S的栈顶，由此往前在栈中找最左素短语的头符号S_{j+1}，直到找到第一个 $<\cdot$ 为止；

3）已找到最左素短语$S_{j+1}\cdots S_k$，将其归约为某个非终结符N及做相应的语义处理。

主控程序：　设 k 为符号栈S 的指针

```
1        k =1;   S[k]:= "#";
2        REPEAT
3          把下一个输入字符读进a中;
4          IF  S[k]∈VT  THEN  j:= k  ELSE  j:=
5          WHILE  S[ j ]> a   DO  #当栈顶算符
6            BEGIN
7              REPEAT
8                  Q:= S[ j];
9                  IF  S[ j -1]∈VT  THEN  j:= j -1   ELSE  j:= j -2
10             UNTIL   S[ j] <Q;
11             把S[ j +1] …S[k] 归约为某个N;
12             k := j+1;     S[k]:= N
13         END OF  WHILE;
14         IF  S[ j] < a  OR  S[ j] = a  THEN
15               BEGIN k := k+1;  S[k]:= a  END
16         ELSE  ERROR
17       UNTIL  a = "#"
```

自左至右，终结符对终结符，非终结符对非终结符，而且对应的终结符相同。

$$N \rightarrow \begin{matrix} X_1 & X_2 & \cdots & X_{k-j} \\ \updownarrow & \updownarrow & & \updownarrow \\ S[j+1] & S[j+2] & \cdots & S[k] \end{matrix}$$

例. 对例1中文法G，符号串i*(i+i)的分析过程如下：

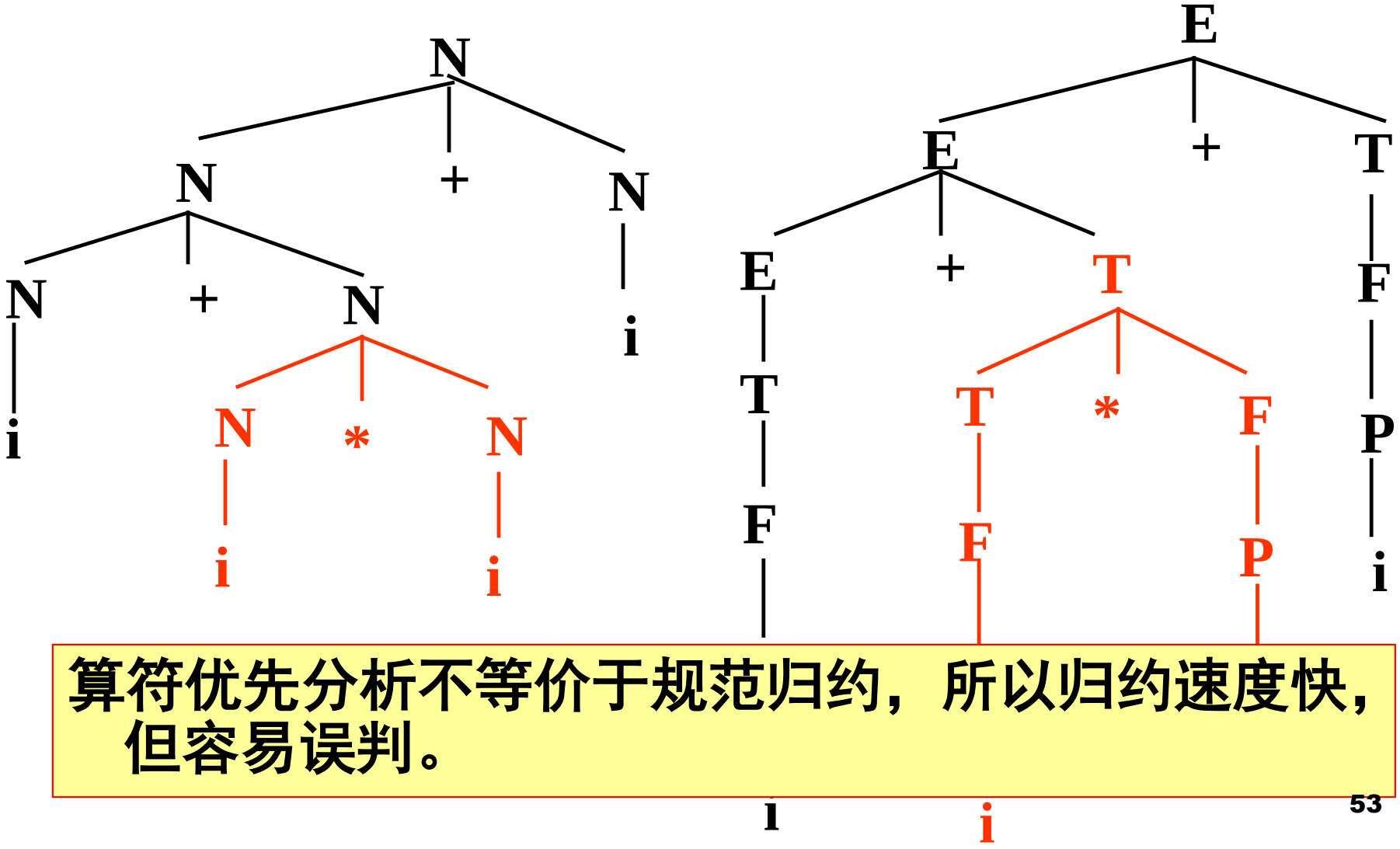
符号栈	关系	输入串	最左素短语
#	<.	i*（i+i）#	
#i	>.	*（i+i）#	i
#N	<.	*（i+i）#	
#N*	<.	（i+i）#	
#N*(	<.	i+i）#	
#N*(i	>.	+i）#	i
#N*(N	<.	+ i）#	
#N*(N+	<.	i）#	
#N*(N+i	>.	）#	i
#N*(N+N	>.	）#	N+N
#N*(N	=.	）#	
#N*(N)	>.	#	（N）
#N*N	>.	#	N*N
#N	=.	#	
#N#		成功	

说明

- 在算法的工作过程中，若出现j减1后的值小于等于0时，则意味着输入串有错。在正确的情况下，算法工作完毕时，符号栈S应呈现：# N #。
- 由于非终结符对归约没有影响，因此，非终结符根本可以不进符号栈S。

例. 对文法G: (1) E→E+T | T　(2) T→T*F | F
(3) F→P↑F | P　(4) P→(E) | i

分别给出句子i+i*i+i的算符优先分析和规范规约分析的语法树



算符优先分析不等价于规范归约，所以归约速度快，但容易误判。

随堂练习

- （Canvas）文法

G: (1) E→E+T | T　　(2) T→T*F | F

　　(3) F→P↑F | P　　(4) P→(E) | i

对句子i↑i+i采用算符优先分析，需要规约多少步，采用规范规约分析，需要规约多少步?

- P133,第3题

内容线索

- ✓ 自下而上分析基本问题
- ✓ 规范规约
- ✓ 算符优先分析方法
- ■ LR分析方法

LR分析法

- 在自下而上的语法分析中，算符优先分析算法只适用于算符优先文法，还有很大一类上下文无关文法可以用LR分析法分析。
- LR分析法中的L表示从左向右扫描输入串， R表示构造最右推导的逆。LR分析法是严格的规范规约。
- 不足：LR分析法手工构造分析程序工作量相当大。
  - YACC是一个语法分析程序的自动生成器。

LR分析法

- LR分析法：1965年 由Donald Knuth（高纳德） 提出

产生分析表

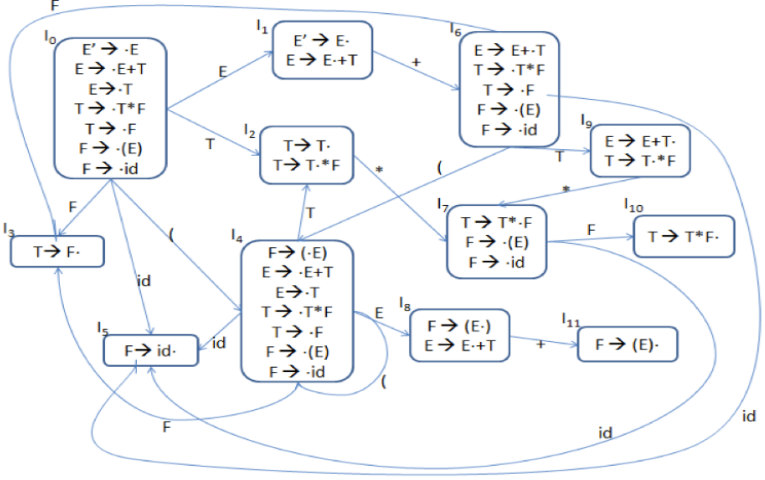
LR分析器工作

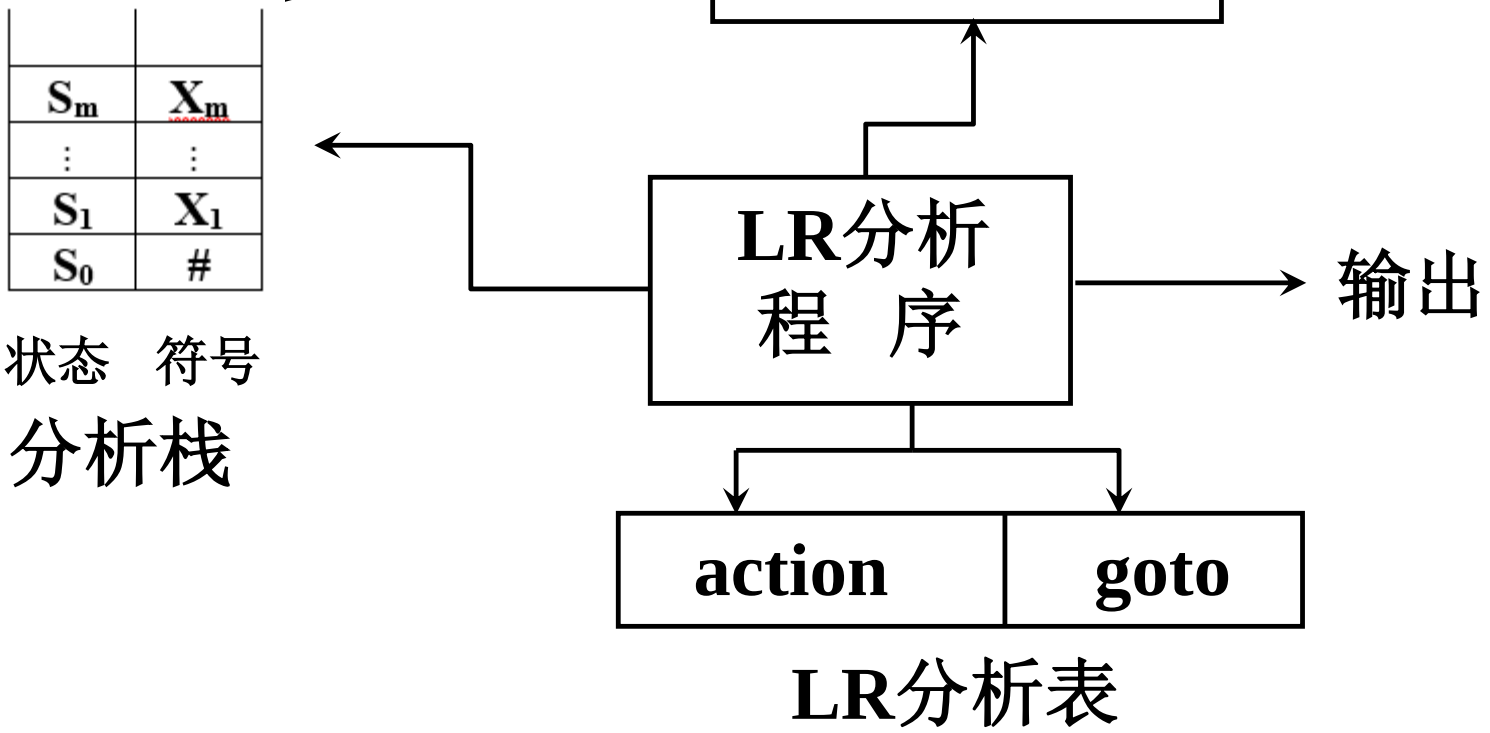
LR分析法原理

- 在移进 - 归约过程中寻找句柄
  - 历史: 在分析栈中已移进和归约的符号串
  - 展望: 根据当前使用的产生式推测未来可能遇到的输入符号
  - 现实: 当前输入符号

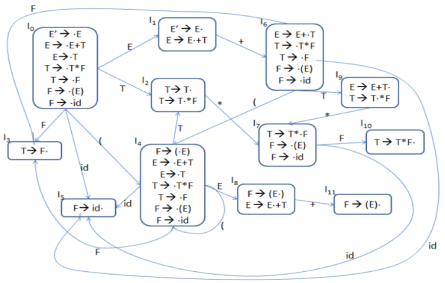
LR分析器模型

把“历史”及“展望”综合抽象成状态；
由栈顶的状态和现行的输入符号唯一确定每一步工作





分析表



- LR分析器的核心是一张分析表
  - ACTION[s, a]：当状态s面临输入符号a时，应采取什么动作.
  - GOTO[s, X]：状态s面对文法符号X时，下一状态是什么
    - GO[s, X]定义了一个以文法符号为字母表的DFA

LR分析法

- 总控程序: 所有的LR分析器相同
- 分析表: 是自动生成语法分析器的关键
  - LR (0) 表：基础、有局限性
  - SLR表：简单LR表，实用
  - 规范LR表：能力强、代价大
  - LALR表：向前LR表，介于SLR和规范LR之间

举个例子

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

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

Action[s, a]

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

- 每一项ACTION[s, a]所规定的四种动作:
  1. **移进** 把(s, a)的下一状态s'和输入符号a推进栈，下一输入符号变成现行输入符号.
  2. **归约** 指用某产生式$A\rightarrow\beta$进行归约. 假若β的长度为r， 归约动作是：去除栈顶r个项，使状态s_{m-r}变成栈顶状态，然后把(s_{m-r}, A)的下一状态s'=GOTO[s_{m-r}, A]和文法符号A推进栈.
  3. **接受** 宣布分析成功，停止分析器工作.
  4. **报错**

分析过程

三元式（栈内状态序列，移进归约串，输入串）的变化：

开　始:　（　S_0,　　#,　　$a_1a_2 \ldots a_n$# ）

某一步:　($S_0S_1 \ldots S_m$,　　$\#X_1X_2 \ldots X_m$,　　$a_i\ a_{i+1} \ldots a_n$ #)

下一步:　ACTION [S_m, a_i]

若 ACTION [S_m, a_i] 为 “移进” 且GOTO [S_m, a_i] = S

则三元式为

（ $S_0S_1 \ldots S_m$ **S**,　　$\#X_1X_2 \ldots X_m$**a_i** ,　　$a_{i+1} \ldots a_n$# ）

若 ACTION [S_m, a_i] 为 “ 归约” { A→β},

且 $|\beta|$ = r ,β= $X_{m-r+1} \ldots X_m$, GOTO [S_{m-r}, A] =S,

则三元式为：

($S_0S_1 \ldots S_{m-r}$ **S**,　　$\#X_1X_2 \ldots X_{m-r}$ **A**,　$a_i\ a_{i+1} \ldots a_n$ # ）

若ACTION [S_m, a_i] 为 “ 接受” 则结束

若ACTION [S_m, a_i] 为 “ 报错” 则进行出错处理

LR语法分析程序

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

Figure 4.36: LR-parsing program

Book: Compilers Principles Techniques and Tools (2nd Edition) P251

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

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

输入串为i*i+i，工作过程:

步骤	状态	符号	输入串
(1)	0	#	i*i+i#
(2)	05	#i	*i+i#
(3)	03	#F	*i+i#
(4)	02	#T	*i+i#
(5)	027	#T*	i+i#
(6)	0275	#T*i	+i#
(7)	02710	#T*F	+i#
(8)	02	#T	+i#
(9)	01	#E	+i#
(10)	016	#E+	i#
(11)	0165	#E+i	#
(12)	0163	#E+F	#
(13)	0169	#E+T	#
(14)	01	#E	#
(15)	接受		

LR文法

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

- LR文法: 对于一个文法，如果能够构造一张LR分析表, 使得它的**每个入口均是唯一确定**，则该文法称为LR文法。
  - 在进行自下而上分析时， 一旦栈顶形成句柄，即可归约。
- LR(k)文法：对于一个文法，如果每步至多向前检查 k个输入符号，就能用LR分析器进行分析。则这个文法就称为LR(k)文法。
  - 大多数程序语言，符合LR(1)文法
- LR(0) 文法：k ＝0，即只要根据当前符号和历史信息进行分析，而无需展望。

LR(0)项目

- **文法G 的产生式右部加一个圆点(·)，称为G的一个LR(0)项目。它指明了在分析过程的某时刻看到产生式的多大部分。**

例. G(E):　E→ aA

A→ bA|a

则G的LR（0）项目有:

E → ·aA　　E→ a·A　　E→ aA·

A → ·bA　　A→ b·A　　A→ bA·

A → ·a　　A →a·

例. 文法G(S′)

S′→E

E→aA|bB

A→cA|d

B→cB|d

该文法的项目有：

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

接下来

对于LR(0)如何构造状态?

活前缀

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

- 前缀: 一个字的任意首部。例:字abc的前缀有ε,a,ab,或abc.
- 活前缀: 规范句型的一个前缀，前缀的尾符号最多包含到句型的句柄，即这种前缀不含句柄之后的任何符号（可归前缀）。
  - 即，对于规范句型αβδ，β为句柄，如果$\alpha\beta=u_1u_2\ldots u_r$，则符号串$u_1u_2\ldots u_i(1\le i\le r)$是αβδ的活前缀。(δ必为终结符串)
- 在LR分析工作过程的任何时候，栈里的文法符号（自栈底向上）应该构成活前缀。
- 对于一个文法G，可以构造一个识别G的所有活前缀有限自动机，并以此构造LR分析表。

方法一：识别活前缀的NFA方法

- 构造识别文法所有活前缀的NFA

  项目1为NFA的唯一初态，任何状态（项目）均认为是NFA的终态（活前缀识别态）

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

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

   则从状态i画一条标志为X_i的有向边到状态j；

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

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

- 把识别文法所有活前缀的NFA确定化。

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

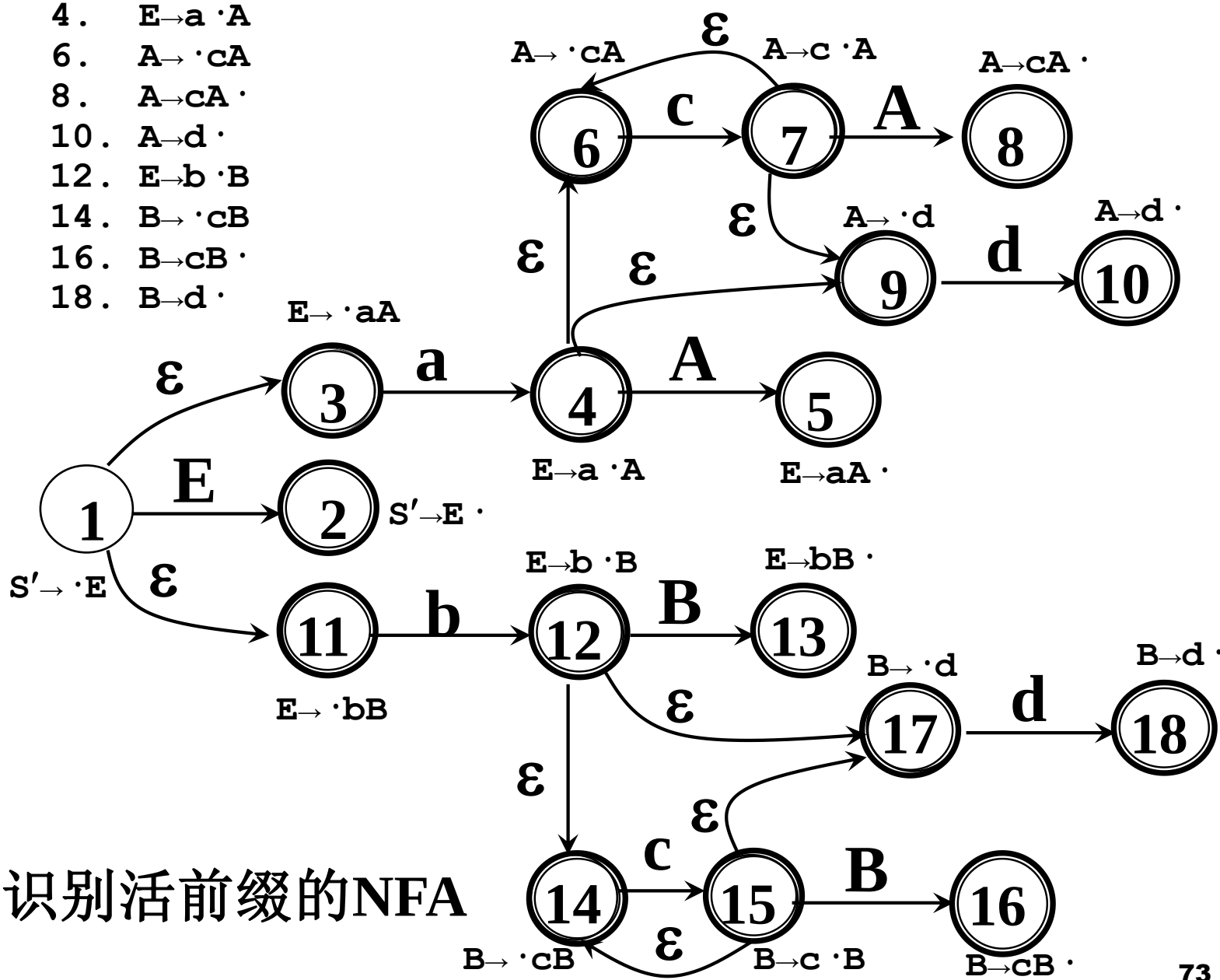
例. 文法G(S′)

S′→E

E→aA|bB

A→cA|d

B→cB|d



识别活前缀的NFA

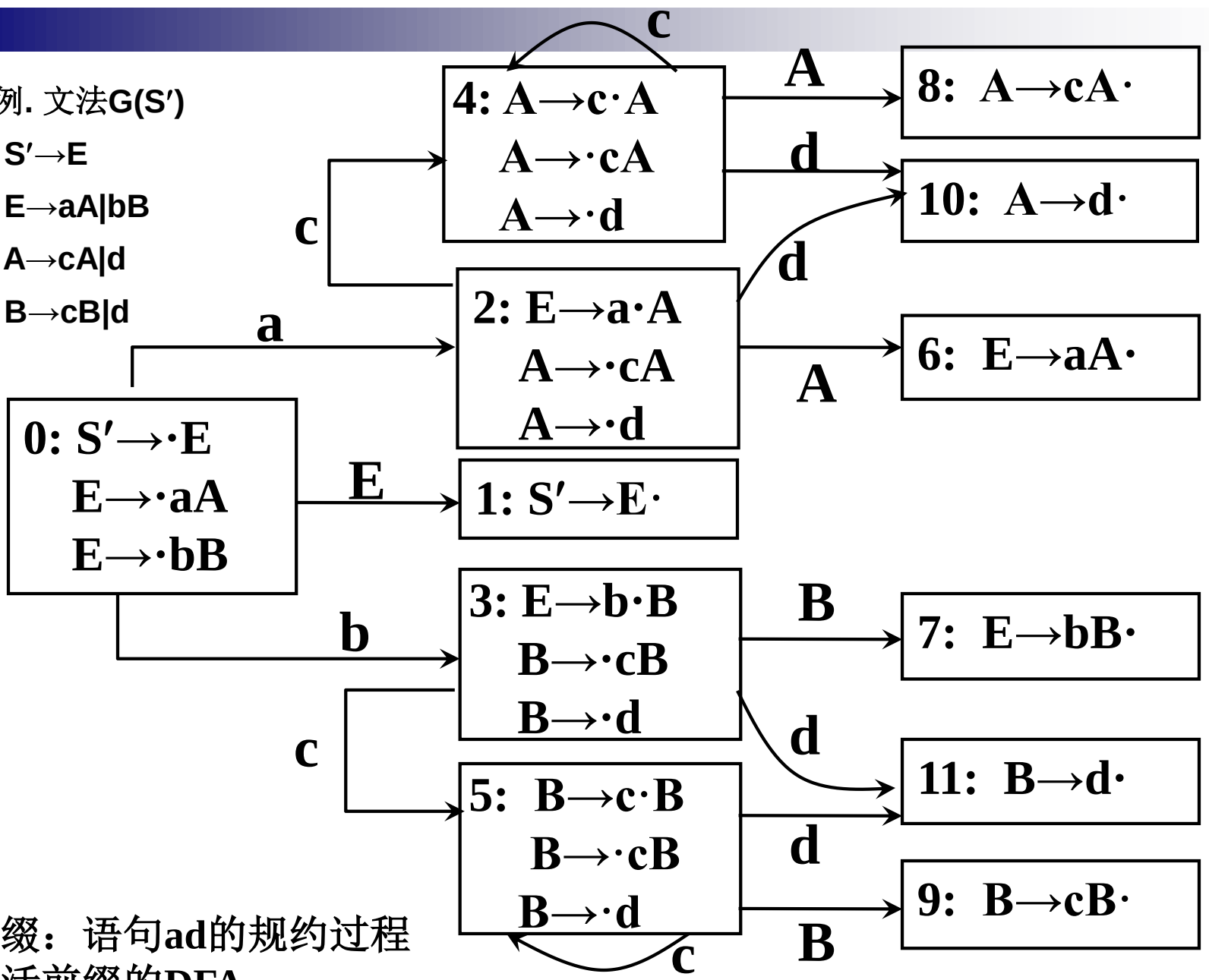
例. 文法G(S′)

S′→E

E→aA|bB

A→cA|d

B→cB|d



活前缀：语句ad的规约过程

识别活前缀的DFA

方法二：LR(0)项目集规范族

- 构成识别一个文法活前缀的DFA的项目集（状态）的全体称为文法的**LR(0)项目集规范族**（canonical LR（0） collection）。
  - $A\rightarrow\alpha\cdot$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”

拓广文法

- 假定文法G是一个以S为开始符号的文法，我们构造一个G′
  - 包含整个G；
  - 引进了一个不出现在G中的非终结符S′（G′的开始符号）；
  - 增加一个新产生式S′→S。

称G′是G的**拓广文法**。

- 拓广文法会有一个仅含项目S′→S·的状态，这就是唯一的“接受”态。

项目集I的闭包CLOSURE(I)

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

  1. I的任何项目都属于CLOSURE(I)；

  2. 若A→α·Bβ属于CLOSURE(I)，那么，对任何关于B的产生式B→γ，项目B→·γ也属于CLOSURE(I)；

  3. 重复执行上述两步骤直至CLOSURE(I) 不再增大为止。

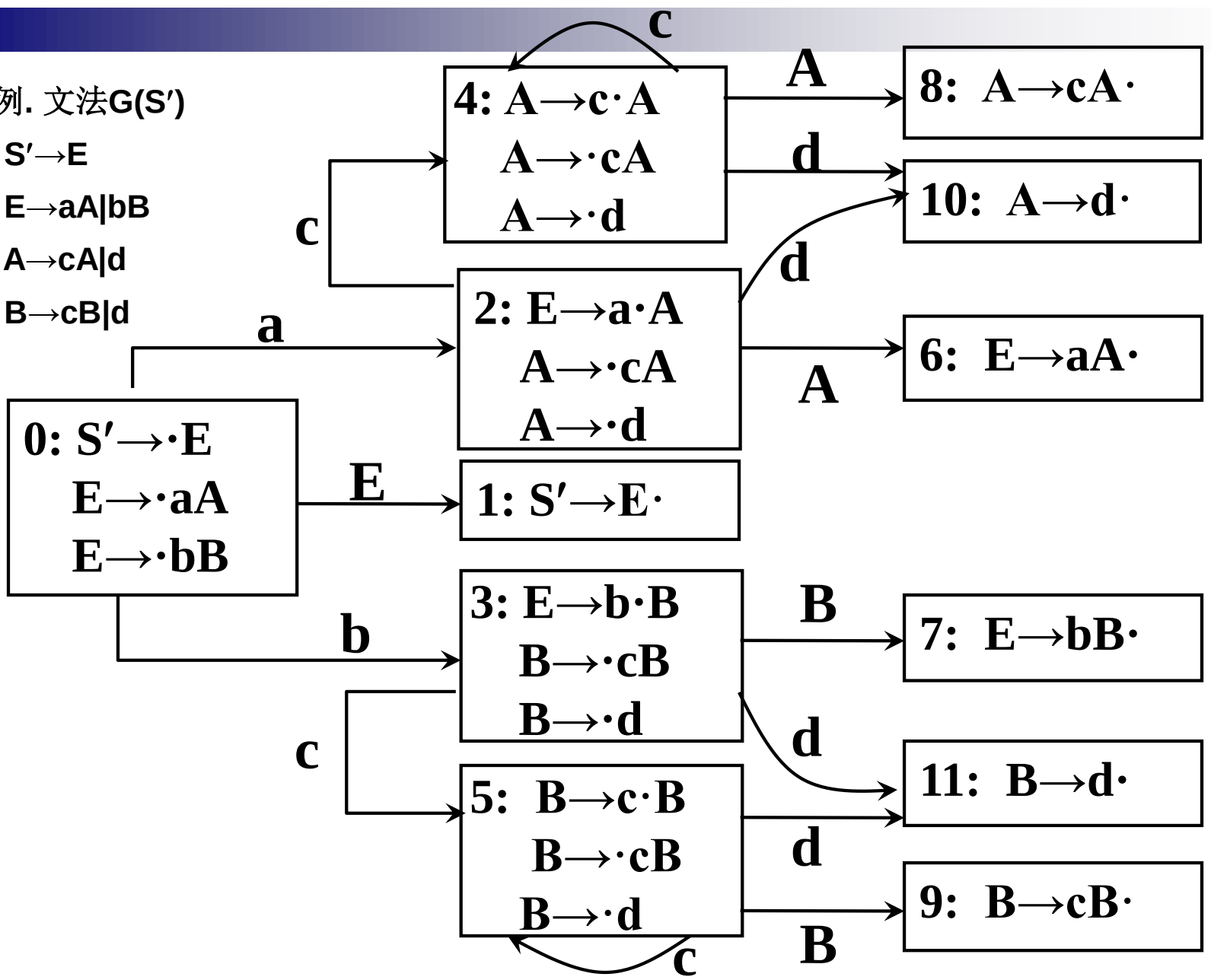
例. 文法G(S′)

S′→E

E→aA|bB

A→cA|d

B→cB|d



状态转换函数GO(I，X)

- GO是一个状态转换函数。I是一个项目集，X是一个文法符号。函数值GO(I，X)定义为：

$$GO(I，X) = CLOSURE(J)$$

其中

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

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。

例.文法G(S′)：S′→E

E→aA|bB

A→cA|d

B→cB|d

设I={S' → ·E}

则CLOSURE(I)={S' → ·E, E→ ·aA, E→ ·bB }

设I_0={S′→·E, E→·aA, E→·bB}

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

= {S' →E·} = I_1

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

= { E→a·A, A→·cA, A→·d}) = I_2

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

= {E →b·B, B→·cB, B→·d} = I_3

LR(0)项目集规范族构造算法

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

- 转换函数GO把项目集连接成一个DFA转换图.

例.文法G(S′)：S′→E

E→aA|bB

A→cA|d

B→cB|d

C=CLOSURE({S′→·E})={{S′→·E, E→. aA, E→. bB }}=I_0

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

= {S’→E·} = I_1

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

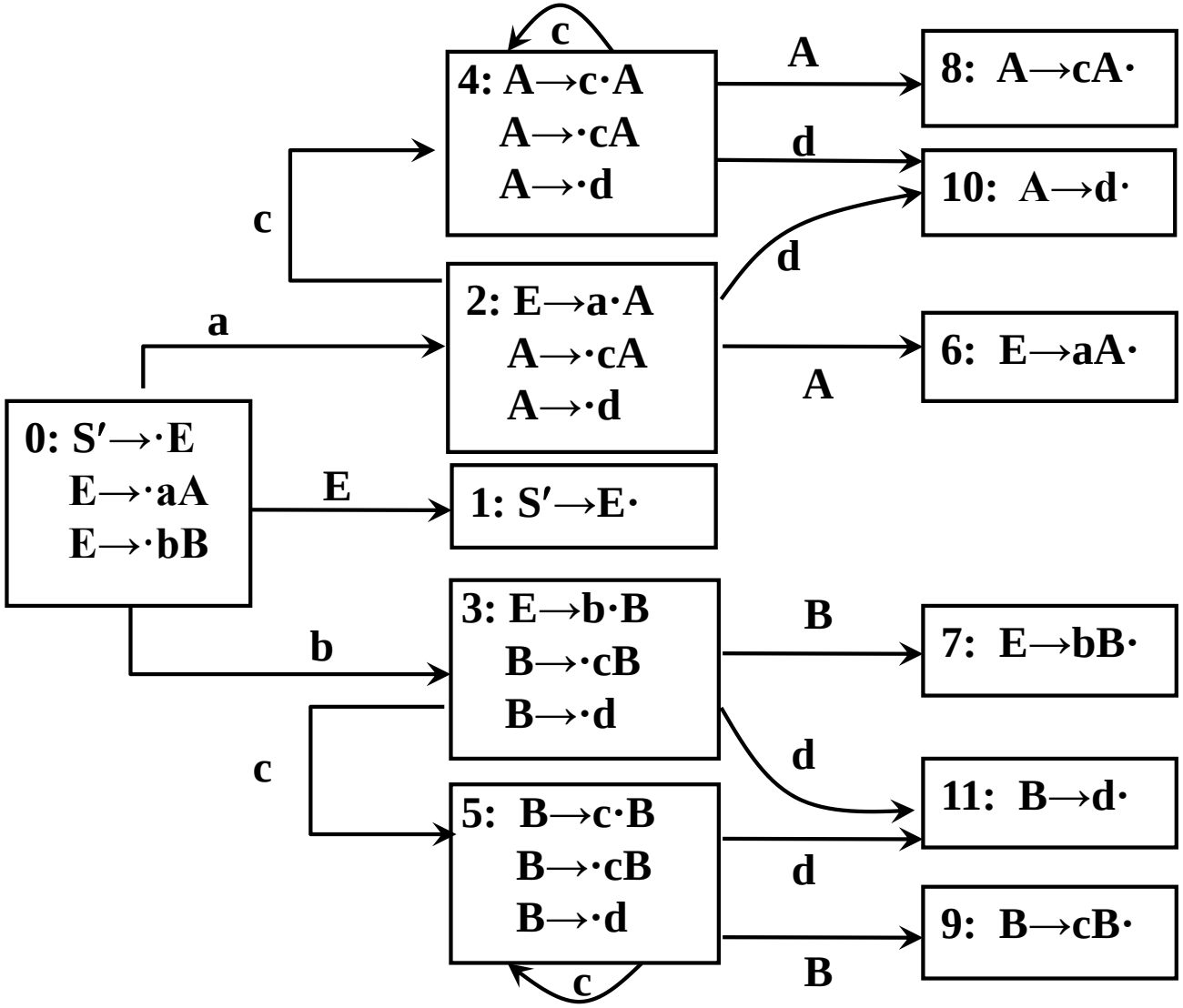
={ E→a·A, A→·cA, A→·d})=I_2

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

={E →b·B, B→·cB, B→·d}= I_3

. . . . . .

可得12个项目集, 即为DFA


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

知识巩固、自测

- **P134 第5题 求构DFA**

5. 考虑文法

$$S \rightarrow AS \mid b$$

$$A \rightarrow SA \mid a$$

(1) 列出这个文法的所有 LR(0)项目。

(2) 构造这个文法的 LR(0)项目集规范族及识别活前缀的 DFA。

(3) 这个文法是 SLR 的吗？若是，构造出它的 SLR 分析表。

(4) 这个文法是 LALR 或 LR(1)的吗？

LR(0)文法

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

  （1） 既含移进项目又含归约项目；

  E->E · *E
  E->E+E ·

  （2）含有多个归约项目

  P->A·
  Q->A·

  则称G是一个LR(0)文法。

  即是：LR(0)文法规范族的每个项目集不包含任何冲突项目

构造LR(0)分析表的算法

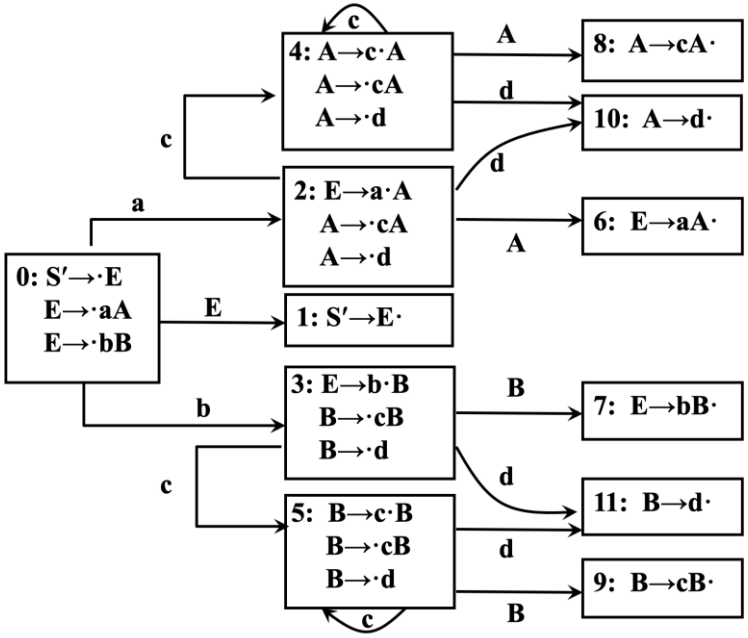
- 令每个项目集I_k的下标k作为分析器的状态
- 包含项目$S' \rightarrow \cdot S$的集合I_k的下标k为分析器的初态。

	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			

分析表的ACTION和GOTO子表构造方法

1. 若项目$A\to\alpha\cdot a\beta$属于I_k且$GO(I_k, a) = I_j$，a为终结符，则置ACTION[k,a] 为： sj
2. 若项目$A\to\alpha\cdot$属于I_k，那么，对任何终结符a(或结束符#)，置ACTION[k,a]为： rj (假定产生式$A\to\alpha$是文法G′的第j个产生式)
3. 若项目$S'\to S\cdot$属于I_k，则置ACTION[k,#]为：acc
4. 若$GO(I_k,A) = I_j$，A为非终结符，则置GOTO[k,A]=j
5. 分析表中凡不能用规则1至4填入信息的空白格均置上：报错标志

文法G(S′)
(0) S′→E
(1) E→aA
(2) E→bB
(3) A→cA
(4) A→d
(5) B→cB
(6) B→d



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

LR分析过程举例

文法G(S′)

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

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

按上表对acccd进行分析

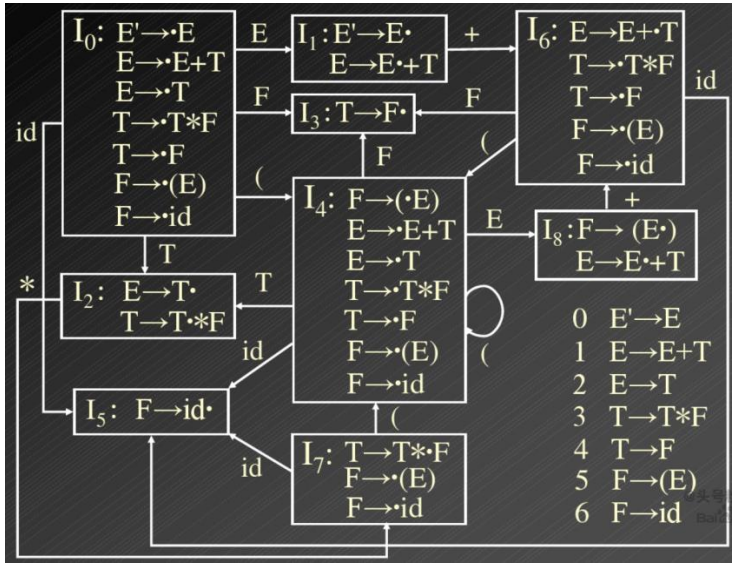
步骤	状态	符号	输入串
1	0	#	acccd#
2	02	#a	cccd#
3	024	#ac	ccd#
4	0244	#acc	cd#
5	02444	#accc	d#
6	0244410	#acccd	#
7	024448	#acccA	#
8	02448	#accA	#
9	0248	#acA	#
10	026	#aA	#
11	01	#E	#

知识巩固、自测

- **Canvas测试**

下面文法G是不是LR(0)文法?

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to \mathbf{id}$



- **对文法G，求LR(0)分析表**

S→E E→E+T|T T→i|(E)

分析语句(i)+i是否合法？ i+i是否合法?

SLR

SLR分析表的构造

- LR(0)文法太简单，没有实用价值.
- 思考：如果项目集中存在移进项目/规约项目，或规约项目/规约项目，是否一定会冲突？ {X→α·bβ，A→α·，B→α·}
- 假定一个LR(0)规范族中含有如下的一个项目集(状态)I＝{X→α·bβ，A→α·，B→α·}。FOLLOW(A)和FOLLOW(B)的交集为∅，且不包含b，那么，当状态I面临任何输入符号a时，可以

  1. 若a=b，则移进；
  2. 若a∈FOLLOW(A)，用产生式A→α进行归约；
  3. 若a∈FOLLOW(B)，用产生式B→α进行归约；
  4. 此外，报错。

SLR分析表的构造

- 假定LR(0)规范族的一个项目集I={$A_1\rightarrow\alpha\cdot a_1\beta_1$，$A_2\rightarrow\alpha\cdot a_2\beta_2$，…，$A_m\rightarrow\alpha\cdot a_m\beta_m$，$B_1\rightarrow\alpha\cdot$，$B_2\rightarrow\alpha\cdot$，…，$B_n\rightarrow\alpha\cdot$} 如果集合{$a_1$，…，$a_m$}，FOLLOW($B_1$)，…，FOLLOW($B_n$)两两不相交(包括不得有两个FOLLOW集合有#)，则
  1. 若a是某个a_i，i=1,2,…,m，则移进；
  2. 若$a\in$FOLLOW(B_i)，i=1,2,…,n，则用产生式$B_i\rightarrow\alpha$进行归约；
  3. 此外，报错。
- 冲突性动作的这种解决办法叫做SLR(1)解决办法。

举例 下面文法的LR(0)项目集规范族为：

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

I_0: S′→·E
E→·E+T
E→·T
T→·T*F
T→·T*F
T→·F
F→· (E)
F→·i

I_1: S′→E·
E→E·+T

I_2: E→T·
T→T·*F

I_3: T→F·

I_4: F→(·E)
E→·E+T
E→·T
T→·T*F
T→·F
F→· (E)
F→·i

I_5: F→i·

I_6: E→E+·T
T→·T*F
T→·F
F→·(E)
F→·i

I_7: T→T*·F
F→·(E)
F→·i

I_8: F→(E·)
E→E·+T

I_9: E→E+T·
T→T·*F

I_{10}: T→T*F·

I_{11}: F→(E)·

- I_1、I_2和I_9都含有“移进 - 归约”冲突

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

FOLLOW(E)＝{#,), +}

$$E \rightarrow T\cdot$$

$$T \rightarrow T\cdot * F$$

因为FOLLOW(*E*) = {#,) ,+}，所以

action[2, #]=*action*[2, +]=*action*[2,)]=*r2*

action[2, ∗] = *s7*

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

状态	ACTION					
	i	+	*	(	)	#
2		r2	s7		r2	r2

其分析表如下:

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

构造SLR(1)分析表方法

- 首先把G拓广为G′，对G′构造LR(0)项目集规范族C和活前缀识别自动机的状态转换函数GO
- 然后使用C和GO，按下面的算法构造SLR分析表：
  - 令每个项目集I_k的下标k作为分析器的状态，包含项目$S'\rightarrow\cdot S$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“sj”
2. 若项目$A\rightarrow\alpha\cdot$属于I_k，那么，对任何终结符a，$a\in FOLLOW(A)$，置ACTION[k,a]为“rj”；其中，假定$A\rightarrow\alpha$为文法G′的第j个产生式
3. 若项目$S'\rightarrow S\cdot$属于I_k，则置ACTION[k,#]为“acc”
4. 若$GO(I_k,A)=I_j$，A为非终结符，则置GOTO[k,A]=j
5. 分析表中凡不能用规则1至4填入信息的空白格均置上“出错标志”

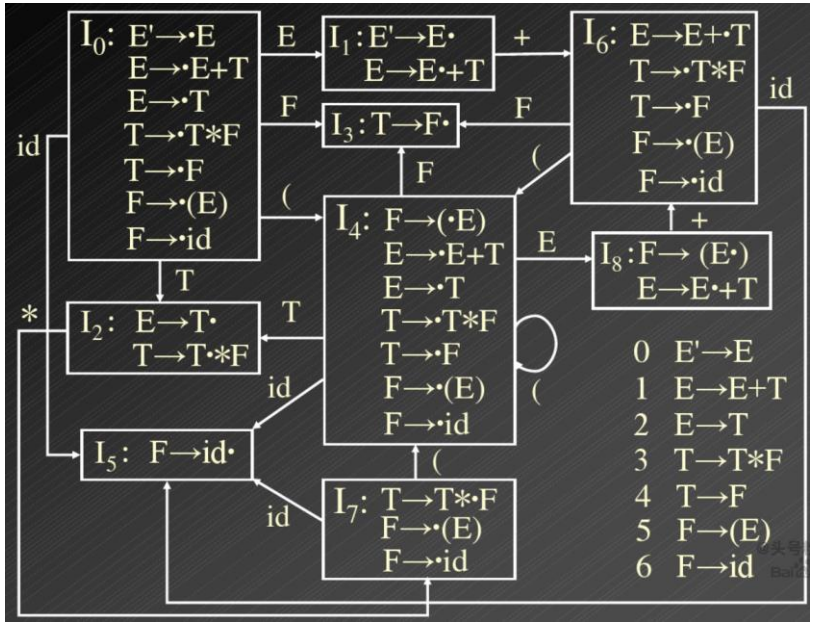
SLR(1)文法

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

自我检查

(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to \mathbf{id}$

- 对文法G1(如上)，求识别活前缀的DFA及LR分析表(P101)，并列出句子id*id的分析过程

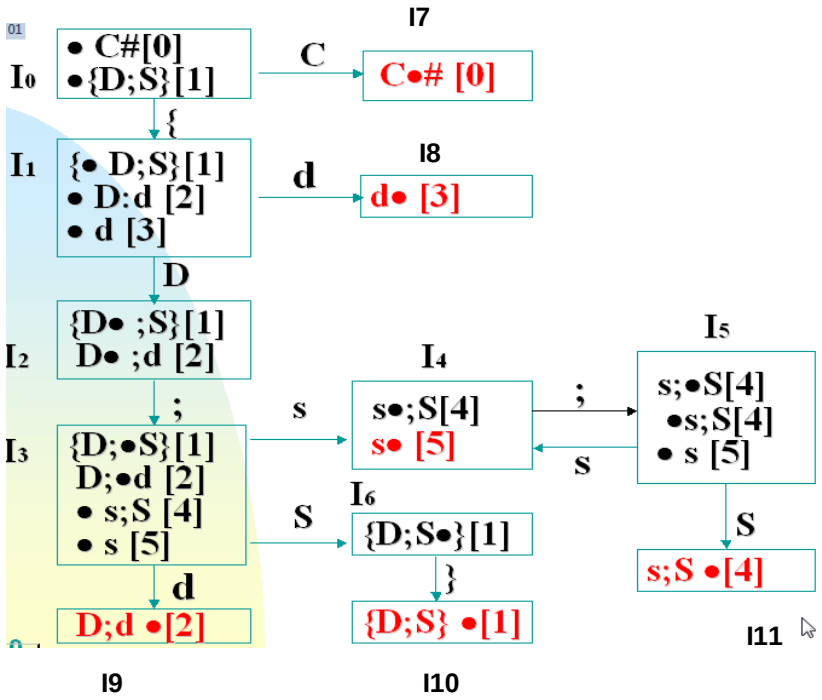


状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

自我检查

- 对文法C’，求SLR(1)分析表，给出字符串{d;s}的分析过程

(0) C’→C
(1) C→{D;S}
(2) D→D;d
(3) D→d
(4) S→s;S
(5) S→s



自我检测

- **A->aAb|ε**
  - 构建LR分析表
  - 分析aabb是否是合法语句

LR(1)

SLR存在的问题：计算FOLLOW集合所得到的超前/后继符号集合可能大于实际能出现的超前/后继符号集。

- **非SLR文法示例：考虑如下文法：**

  (0) S′→S

  (1) S→L=R

  (2) S→R

  (3) L→*R

  (4) L→i

  (5) R→L

(0) S′→S
(1) S→L=R
(2) S→R
(3) L→*R
(4) L→i
(5) R→L

这个文法的LR(0)项目集规范族为：

I_0： S′→·S
S→·L=R
S→·R
L→·*R
L→·i
R→·L

I_1： S′→S·

I_2： S→L·=R
R→L·

I_3： S→R·

I_4： L→*·R
R→·L
L→·*R
L→·i

I_5： L→i·

I_6： S→L=·R
R→·L
L→·*R
L→·i

I_7： L→*R·

I_8： R→L·

I_9： S→L=R·

I_2有“移进－归约”冲突
FOLLOW(R)={#, =}

考虑如下文法：

(0) S′→S

(1) S→L=R

(2) S→R

(3) L→*R

(4) L→i

(5) R→L

不含“R=”为前缀的规范句型
有含“*R=”为前缀的规范句型

I_2：S→L·=R
R→L·

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

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

- FOLLOW集合提供的信息太泛！

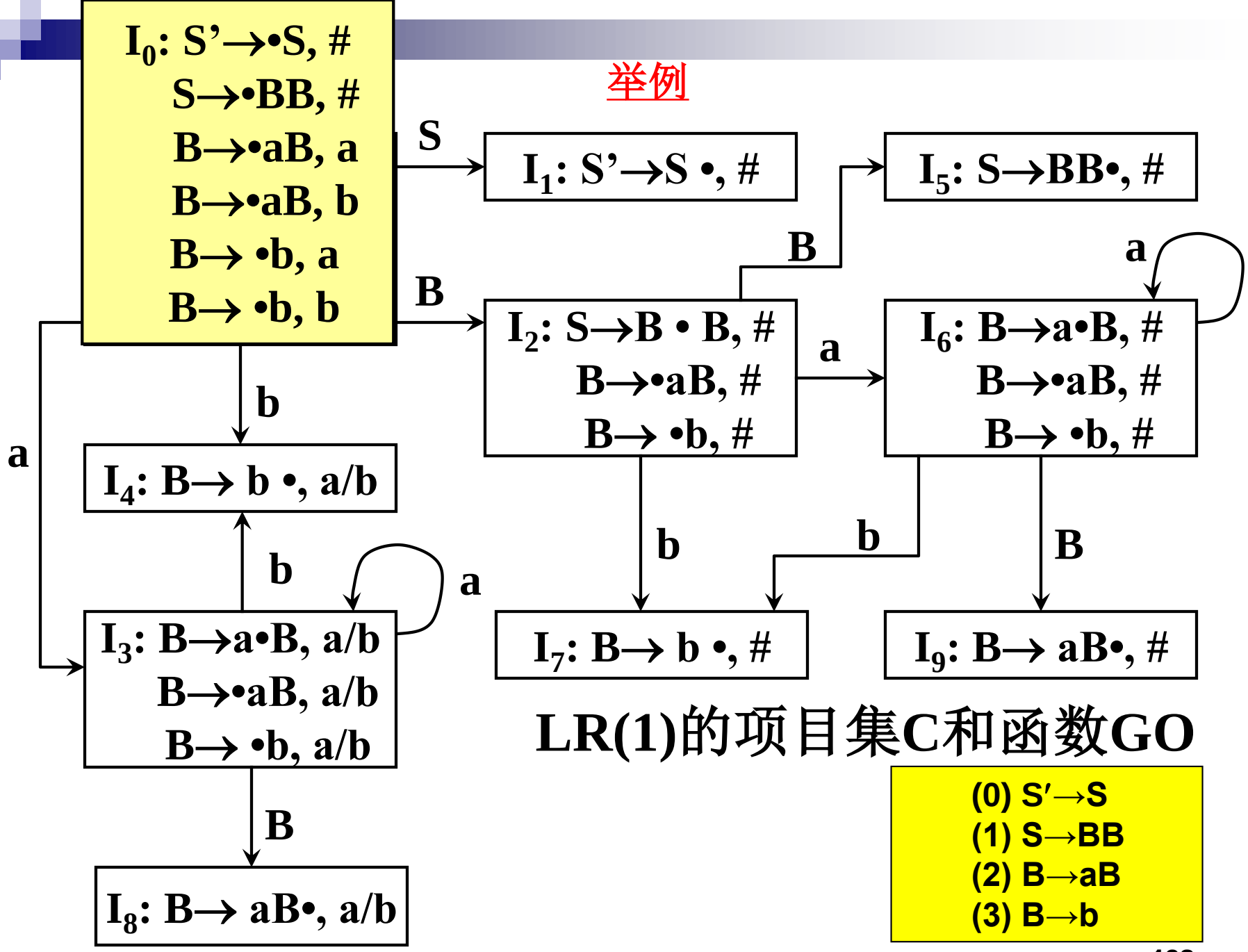
规范LR分析表的构造

- 我们需要重新定义项目，使得每个项目都附带有k个终结符。每个项目的一般形式是$[A\rightarrow\alpha\cdot\beta, a_1a_2\ldots a_k]$，这样的一个项目称为一个**LR(k)项目**。项目中的$a_1a_2\ldots a_k$称为它的**向前搜索符串**(或**展望串**)。
- 向前搜索符串仅对**归约项目**$[A\rightarrow\alpha\cdot, a_1a_2\ldots a_k]$有意义。对于任何**移进或待约项目**$[A\rightarrow\alpha\cdot\beta, a_1a_2\ldots a_k]$, $\beta\neq\varepsilon$，搜索符串$a_1a_2\ldots a_k$没有作用。

- 归约项目[A→α·, $a_1a_2\ldots a_k$]意味着：当它所属的状态呈现在栈顶且后续的k个输入符号为 $a_1a_2\ldots a_k$ 时，才可以把栈顶上的α归约为A。
- 我们只对k≤1的情形感兴趣，向前搜索(展望)一个符号就多半可以确定“移进”或“归约”。
- 形式上我们说一个LR(1)项目[A→α·β, a]对于活前缀γ是**有效的**，如果存在规范推导

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

其中，1) γ＝δα；2) a是ω的 第一个符号，或者a为#而ω为ε。


举例
I0: S'→•S, #
S→•BB, #
B→•aB, a
B→•aB, b
B→ •b, a
B→ •b, b
S
I1: S'→S •, #
I5: S→BB•, #
B
B
a
I2: S→B • B, #
B→•aB, #
B→ •b, #
a
I6: B→a•B, #
B→•aB, #
B→ •b, #
b
a
I4: B→ b •, a/b
b
b
B
b
a
I3: B→a•B, a/b
B→•aB, a/b
B→ •b, a/b
I7: B→ b •, #
I9: B→ aB•, #
B
I8: B→ aB•, a/b
LR(1)的项目集C和函数GO
(0) S'→S
(1) S→BB
(2) B→aB
(3) B→b

LR(1)分析法

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

3. 重复执行步骤2，直至CLOSURE(I)不再增大为止。

- **令I是一个项目集，X是一个文法符号，函数GO(I，X)定义为：**

$$GO(I，X) = CLOSURE(J)$$

其中

J = {任何形如[A→αX·β, *a*]的项目 | [A→α·Xβ, *a*]∈I}

- 文法G′的LR(1)项目集族C的构造算法：

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

LR(1)分析表构造算法

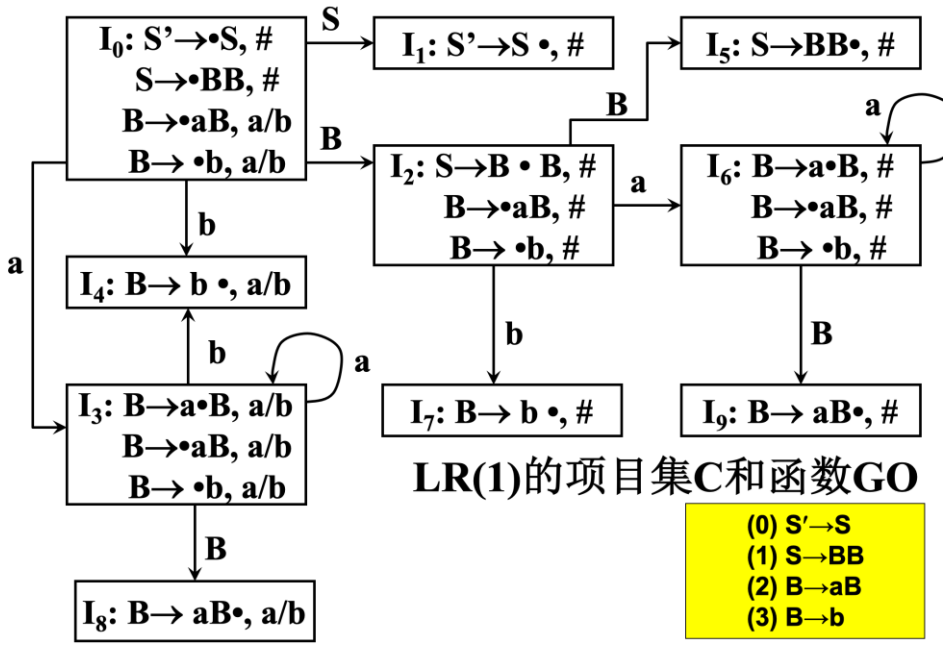
- 构造LR(1)分析表的算法。
  - 令每个I_k的下标k为分析表的状态，令含有[S′→·S, #]的I_k的k为分析器的初态
- 动作ACTION子表和状态转换GOTO子表构造如下：

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

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

  LR(0)⊂SLR ⊂ LR(1) ⊂无二义文法

LR(1)分析表为:



	ACTION			GOTO	
状态	a	b	#	S	B
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

- 按上表对aabab进行分析

步骤	状态	符号	输入串
0	0	#	aabab#
1	03	#a	abab#
2	033	#aa	bab#
3	0334	#aab	ab#
4	0338	#aaB	ab#
5	038	#aB	ab#
6	02	#B	ab#
7	026	#Ba	b#
8	0267	#Baa	#
9	0269	#BaB	#
10	025	#BB	#
11	01	#S	#
	acc		

	ACTION			GOTO	
状态	a	b	#	S	B
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

测试

- Canvas

对于文法S'(赋值表达式)，求其LR(1)分析表

(0) S′→S

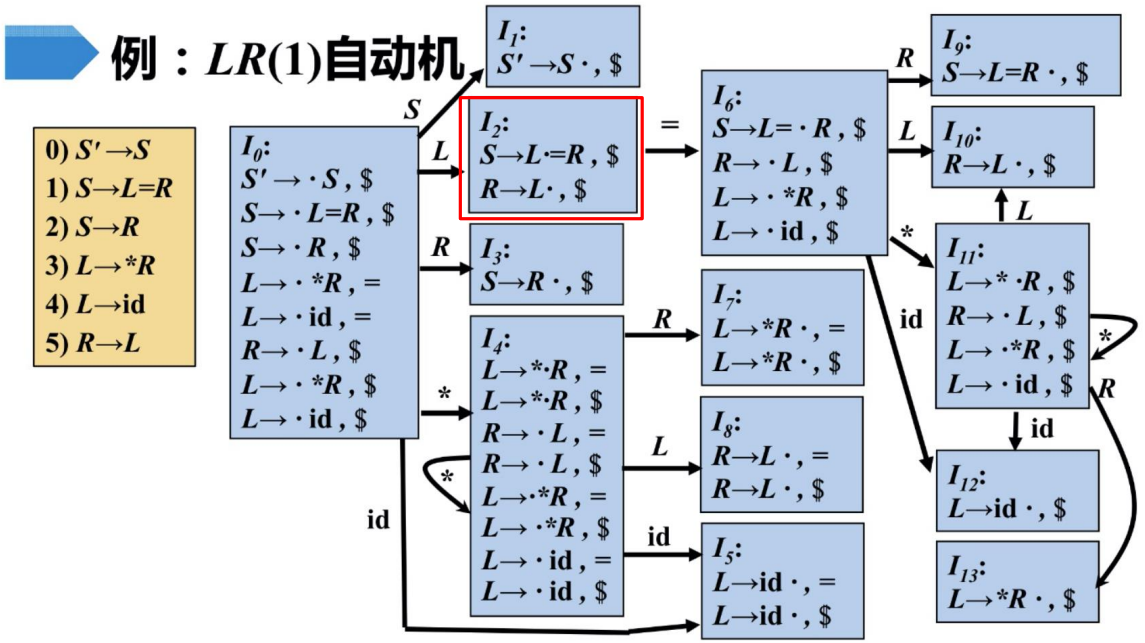
(1) S→L=R

(2) S→R

(3) L→*R

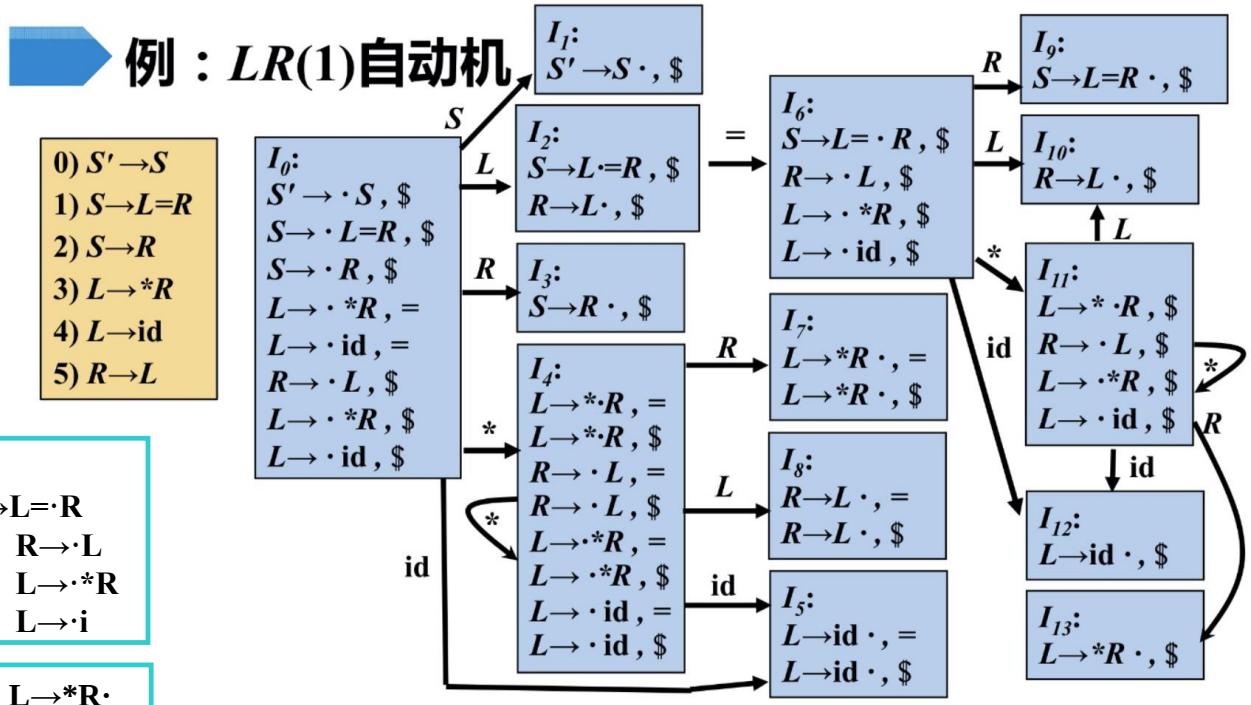
(4) L→id

(5) R→L



例：LR(1)自动机

0) $S' \rightarrow S$
1) $S \rightarrow L=R$
2) $S \rightarrow R$
3) $L \rightarrow *R$
4) $L \rightarrow id$
5) $R \rightarrow L$



LR(0)自动机

I_0： S′→·S
S→·L=R
S→·R
L→·*R
L→·i
R→·L

I_1： S′→S·

I_2： S→L·=R
R→L·

I_3： S→R·

I_4： L→*·R
R→·L
L→·*R
L→·i

I_5： L→i·

I_6：
S→L=·R
R→·L
L→·*R
L→·i

I_7： L→*R·

I_8： R→L·

I_9： S→L=R·

LALR

构造LALR分析表

- LALR分析表构造方法
  - 通过合并规范LR(1)项目集来得到，即对LR自动机进行同心状态合并
- 研究LALR的原因

  规范LR分析表的状态数偏多
- LALR特点
  - LALR和SLR的分析表有同样多的状态，比规范LR分析表要小得多
  - LALR的能力介于SLR和规范LR之间
  - LALR的能力在很多情况下已经够用

LALR的重要特点

(1) LALR分析表和SLR分析表具有相同数目的状态

(2)合并同心状态不会引入新的 移进-规约 冲突，但可能引入新的 规约-规约 冲突

(3) 遇到错误时可能会执行一些多余的规约，但不会执行新的移进

LALR特征(2)

1、合并同心项目集可能会引起冲突

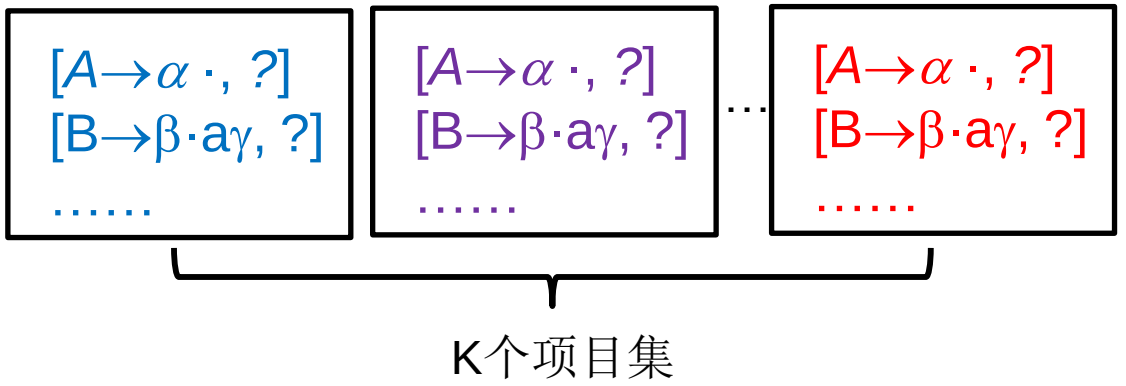
- 同心集的合并不会引起新的移进–归约冲突

合并后项目集

$[A\rightarrow\alpha\cdot, a]$
$[B\rightarrow\beta\cdot a\gamma, ?]$

. . .

合并前项目集



则合并前就有冲突

LALR特征(2)

1、合并同心项目集可能会引起冲突

- 同心集的合并不会引起新的移进–归约冲突
- 同心集的合并有可能产生新的归约–归约冲突

$S' \to S$

$S \to aAd \mid bBd \mid aBe \mid bAe$

$A \to c$

$B \to c$

对ac有效的项目集

$A \to c \cdot, d$

$B \to c \cdot, e$

对bc有效的项目集

$A \to c \cdot, e$

$B \to c \cdot, d$

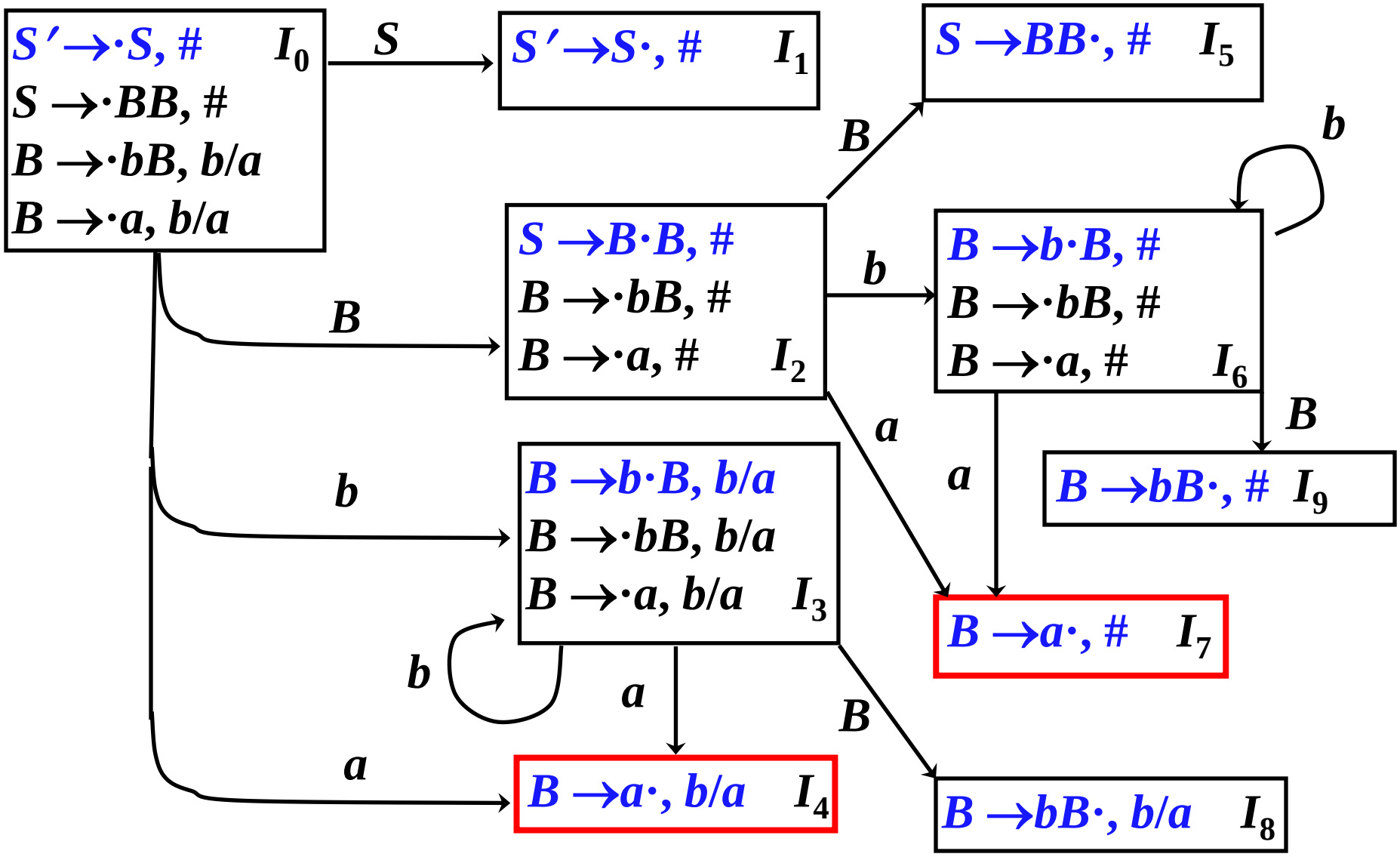
合并同心集后

$A \to c \cdot, d/e$

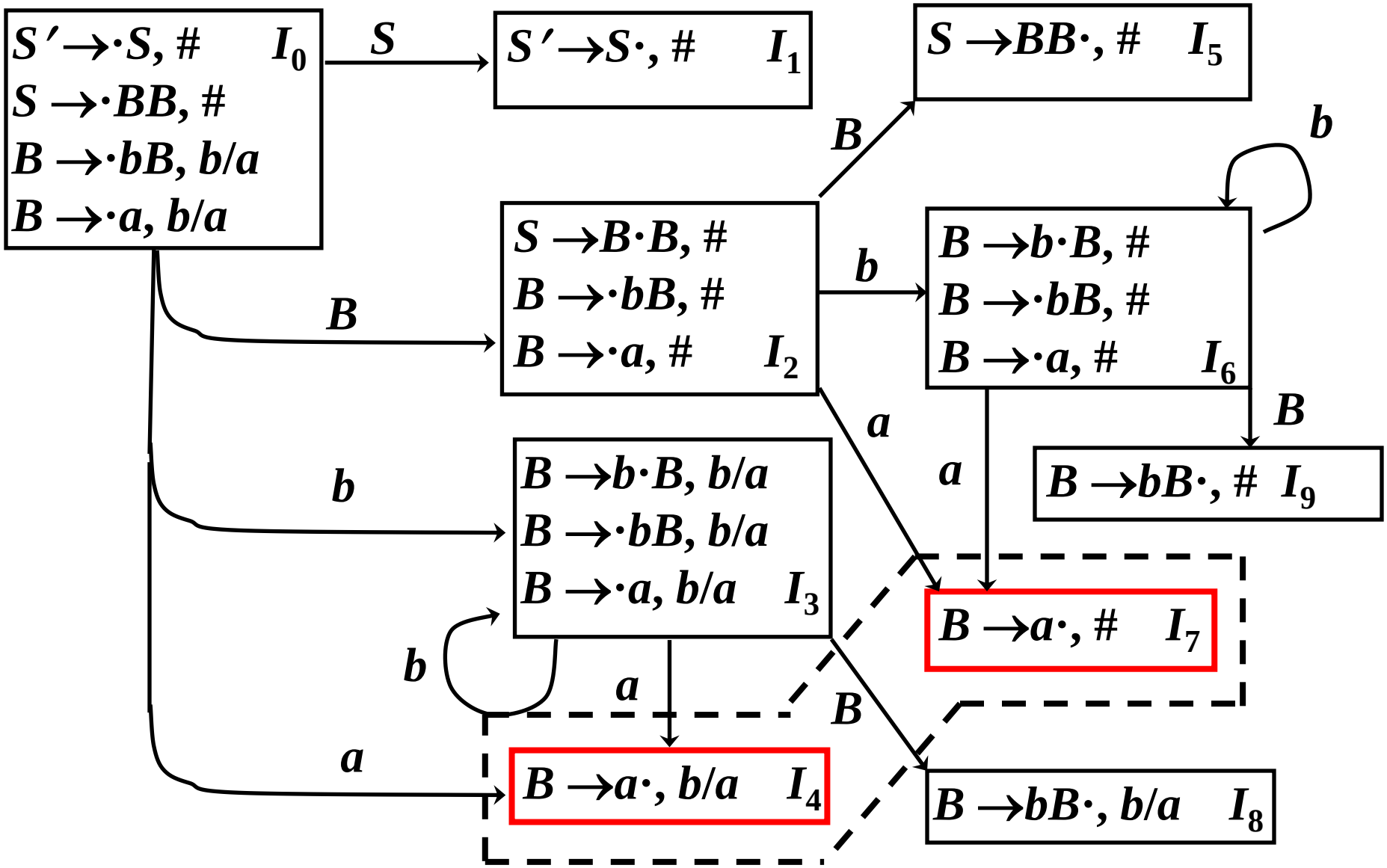
$B \to c \cdot, d/e$

该文法是LR(1)的，但不是LALR(1)的

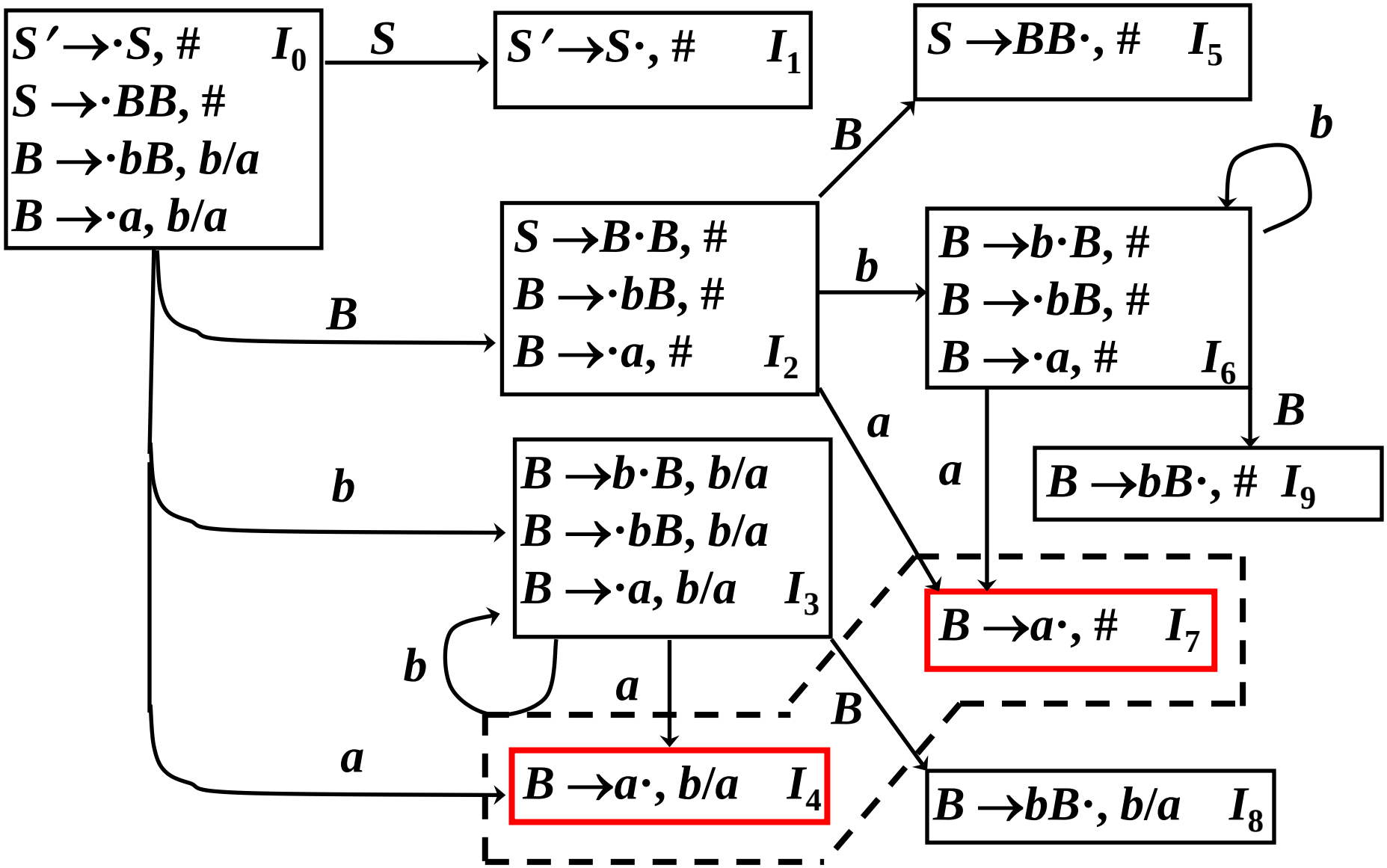
LALR特征(3) I_4和I_7仅搜索符不一样



LALR特征(3)　I_4和I_7合并

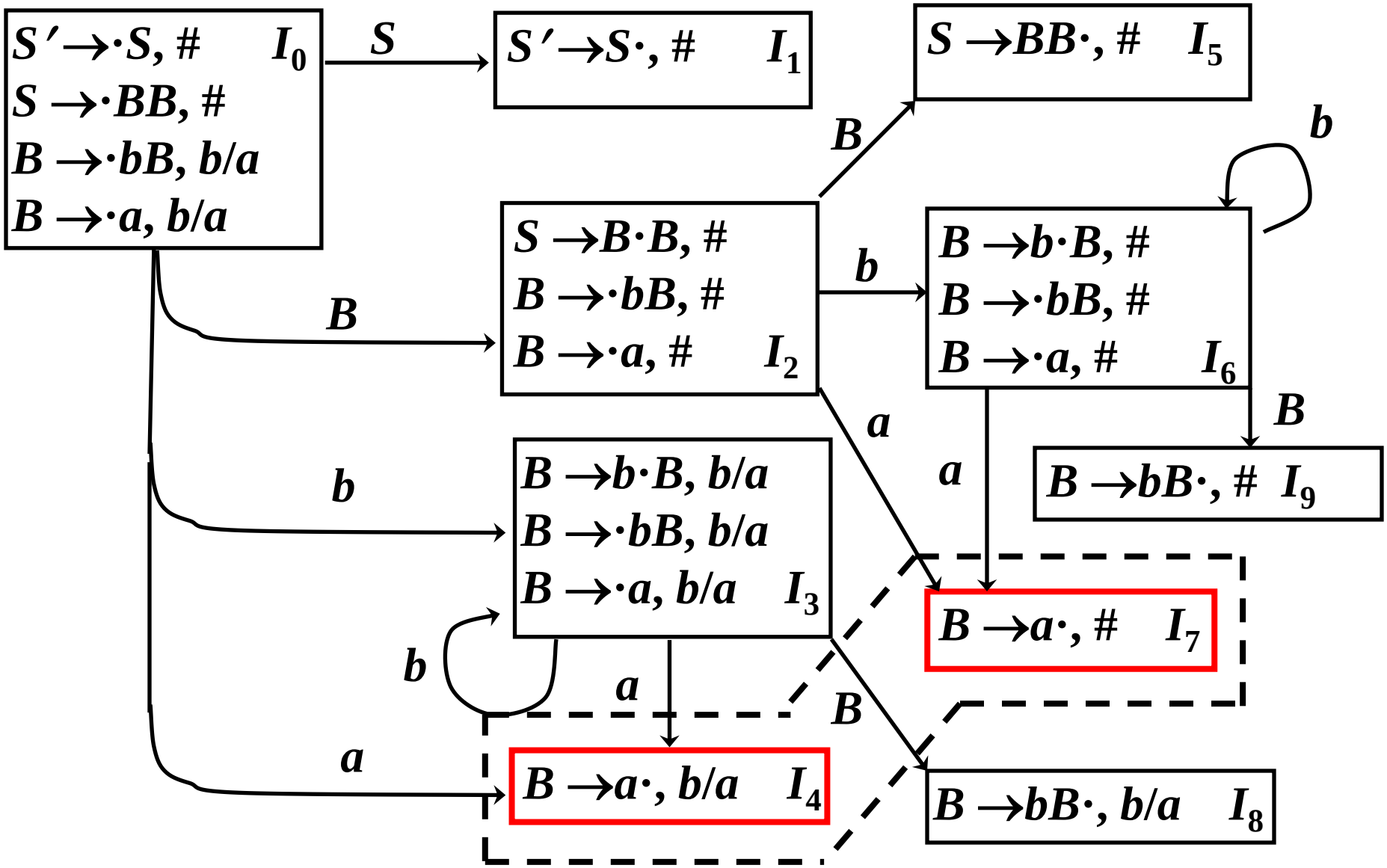


LALR特征(3) 输入为bbabba# 正确

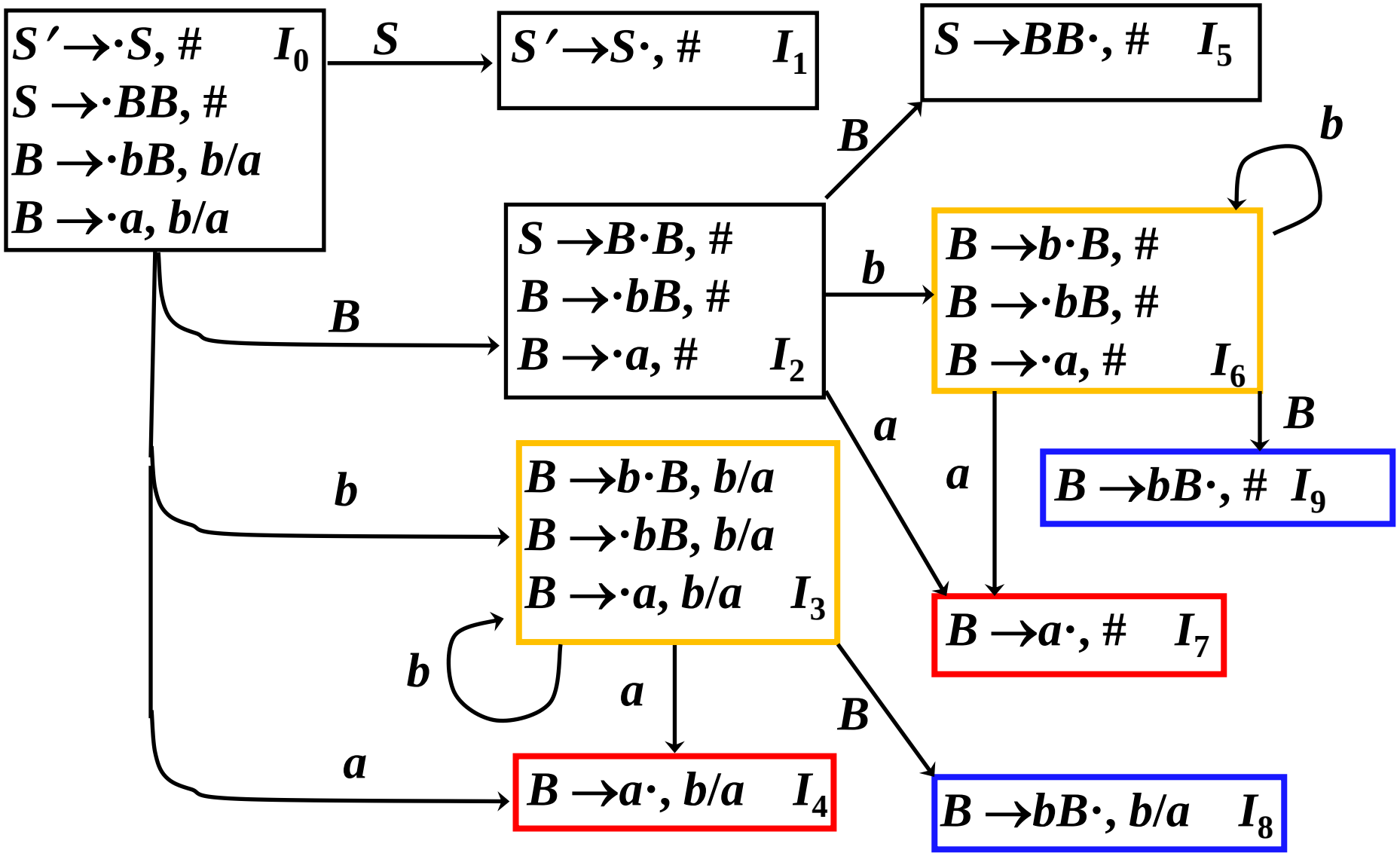


LALR特征(3)

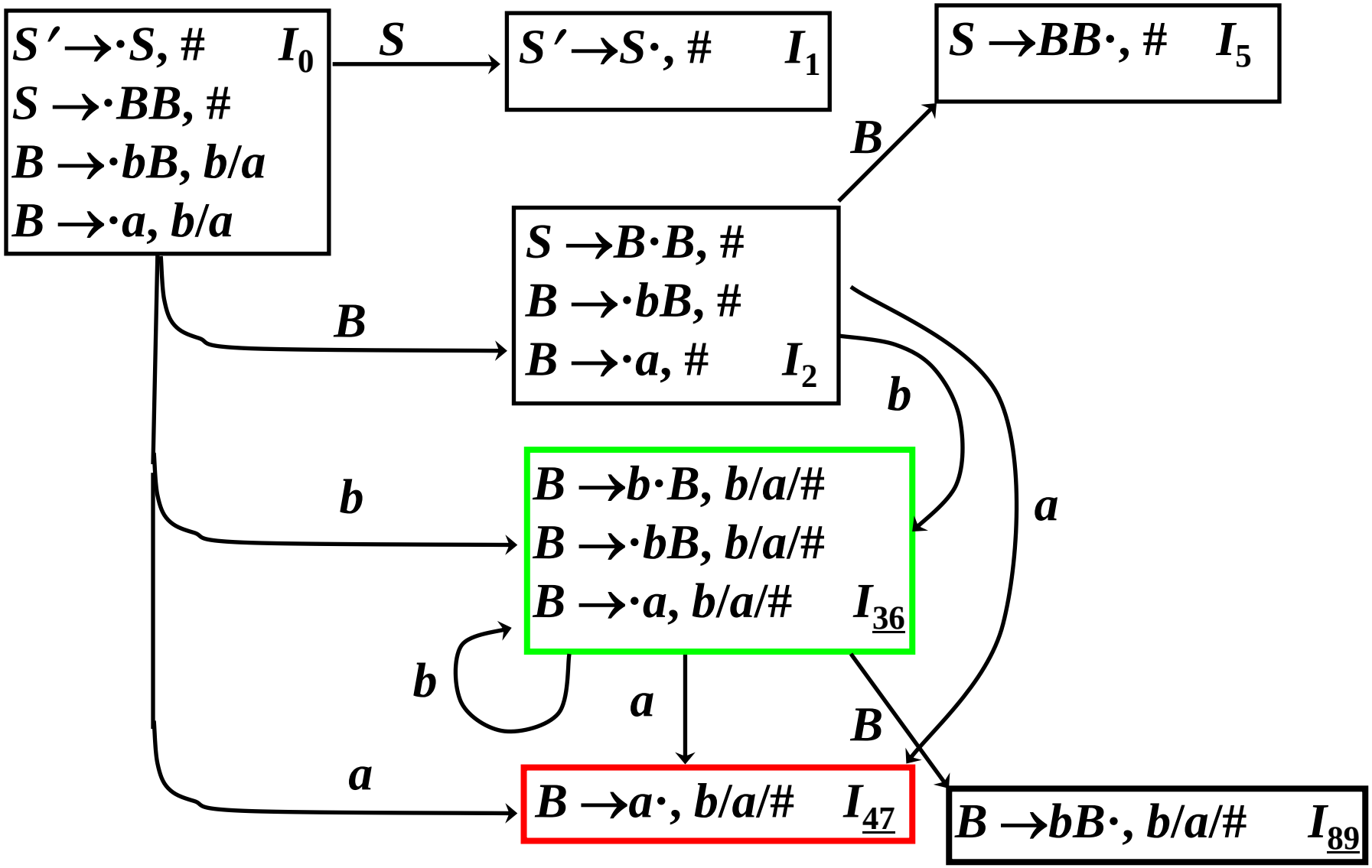
输入为*bba#* 及时报错



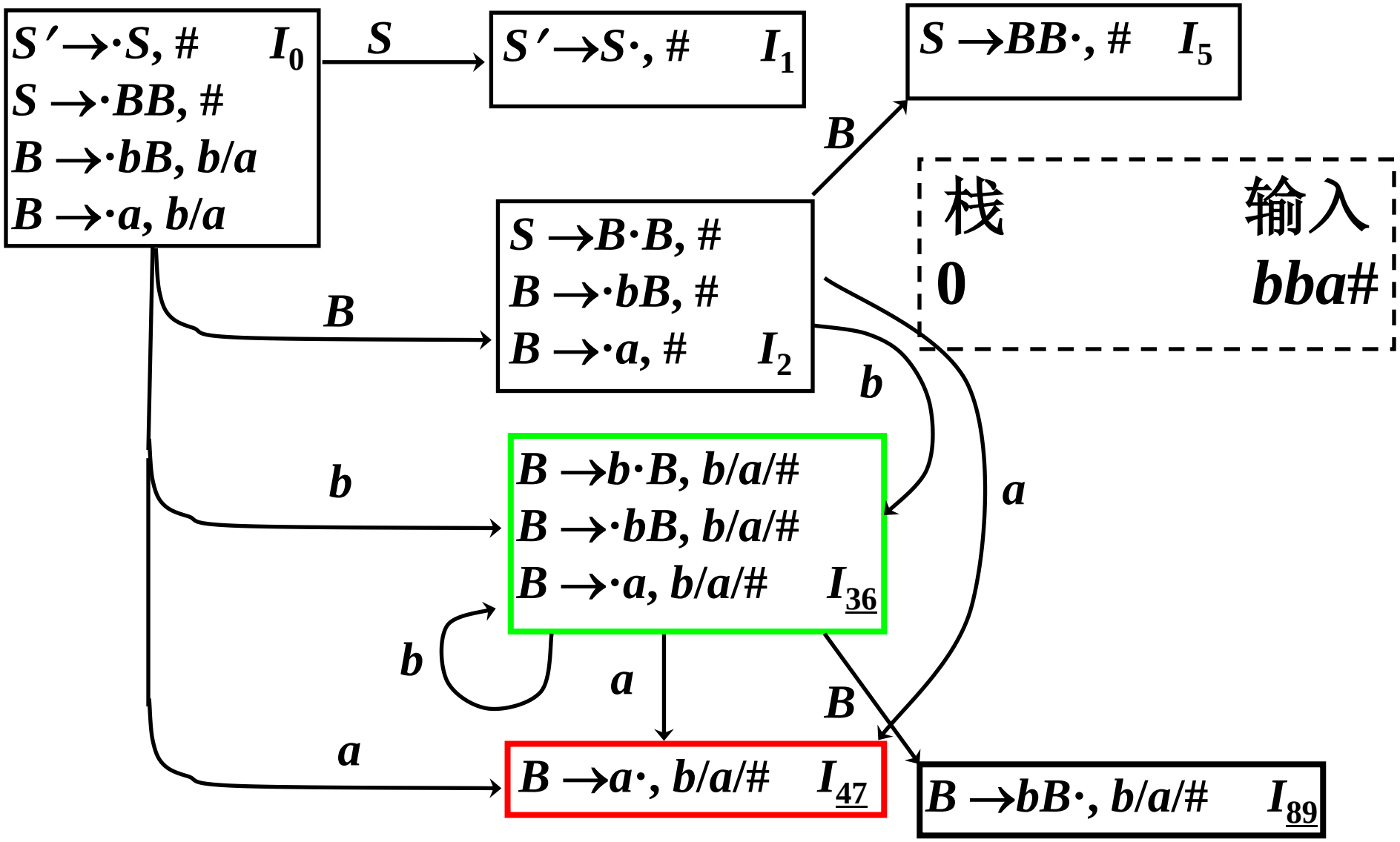
LALR特征(3) 有三组同心集，都合并



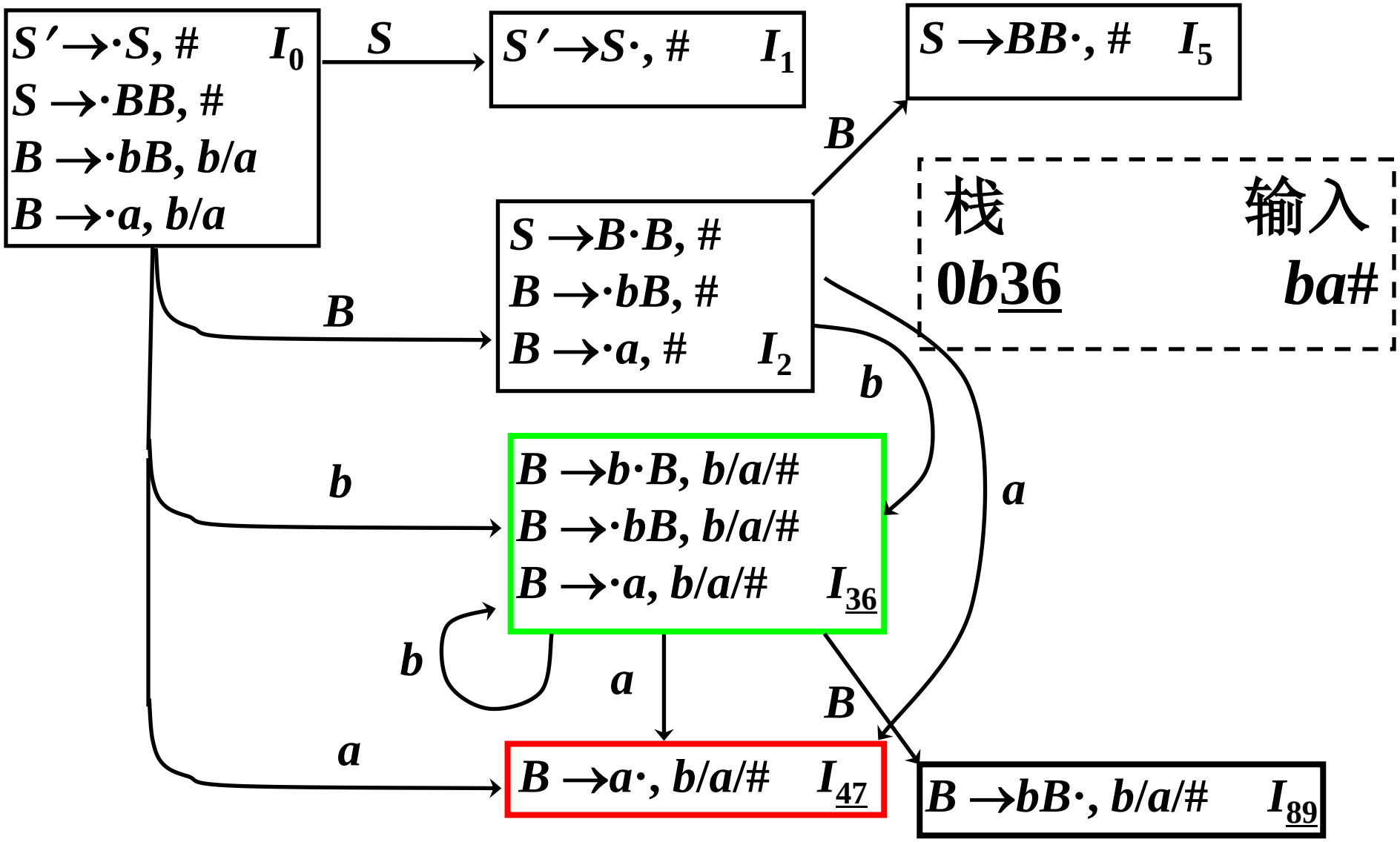
LALR特征(3)


S′→·S, #
S →·BB, #
B →·bB, b/a
B →·a, b/a
I0
S
S′→S·, #
I1
S →BB·, #
I5
B
S →B·B, #
B →·bB, #
B →·a, #
I2
B
b
B →b·B, b/a/#
B →·bB, b/a/#
B →·a, b/a/#
I36
b
a
B
a
B →a·, b/a/#
I47
B →bB·, b/a/#
I89

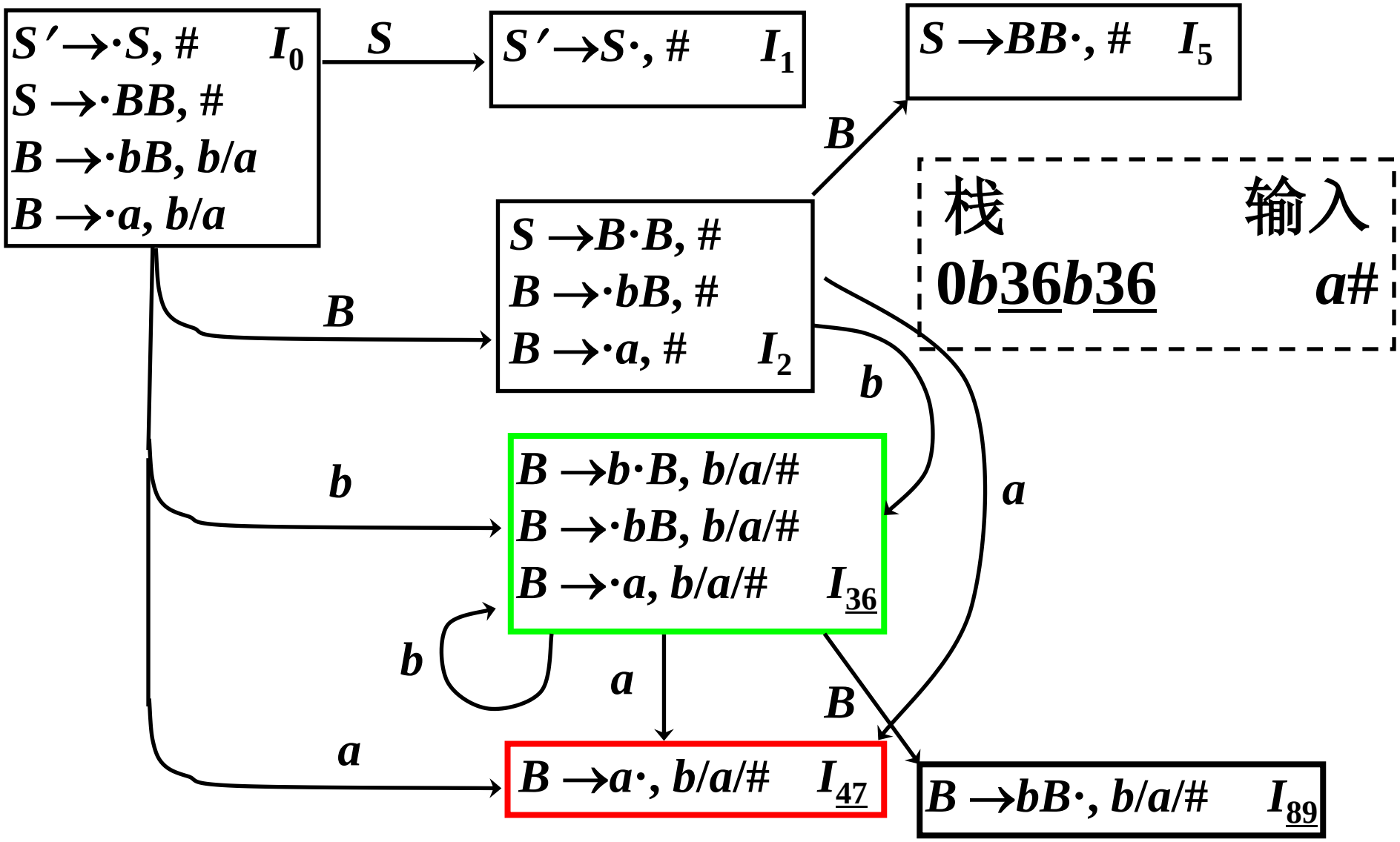
LALR特征(3)


S′→·S, #
S →·BB, #
B →·bB, b/a
B →·a, b/a
I0
S
S′→S·, #
I1
S →BB·, #
I5
B
栈
输入
0
bba#
S →B·B, #
B →·bB, #
B →·a, #
I2
B
b
a
B →b·B, b/a/#
B →·bB, b/a/#
B →·a, b/a/#
I36
b
b
a
B
a
B →a·, b/a/#
I47
B →bB·, b/a/#
I89

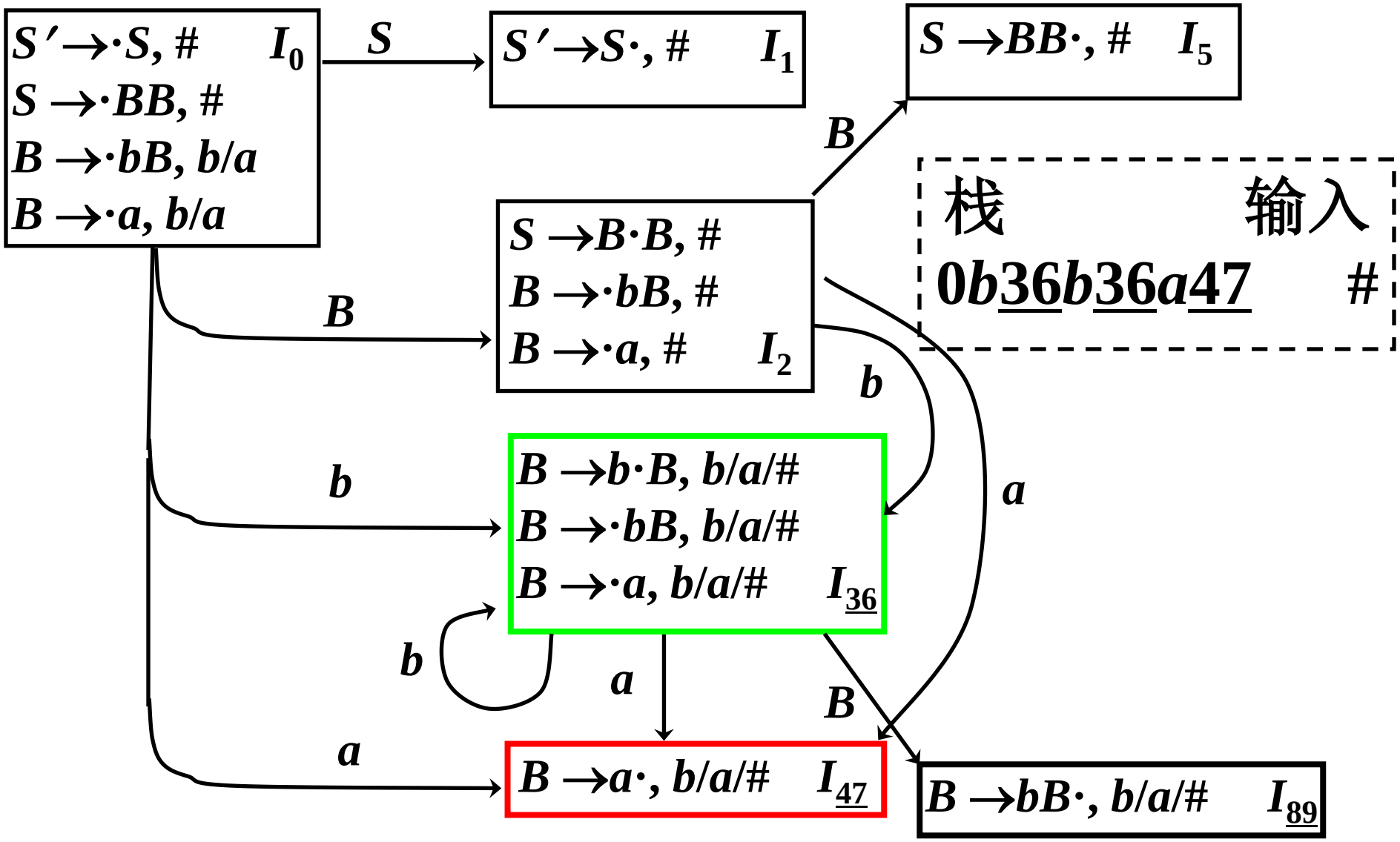
LALR特征(3)


S′→·S, #
S →·BB, #
B →·bB, b/a
B →·a, b/a
I0
S
S′→S·, #
I1
S →BB·, #
I5
B
栈
输入
0b36
ba#
S →B·B, #
B →·bB, #
B →·a, #
I2
B
b
B →b·B, b/a/#
B →·bB, b/a/#
B →·a, b/a/#
I36
b
a
b
a
B
a
B →a·, b/a/#
I47
B →bB·, b/a/#
I89

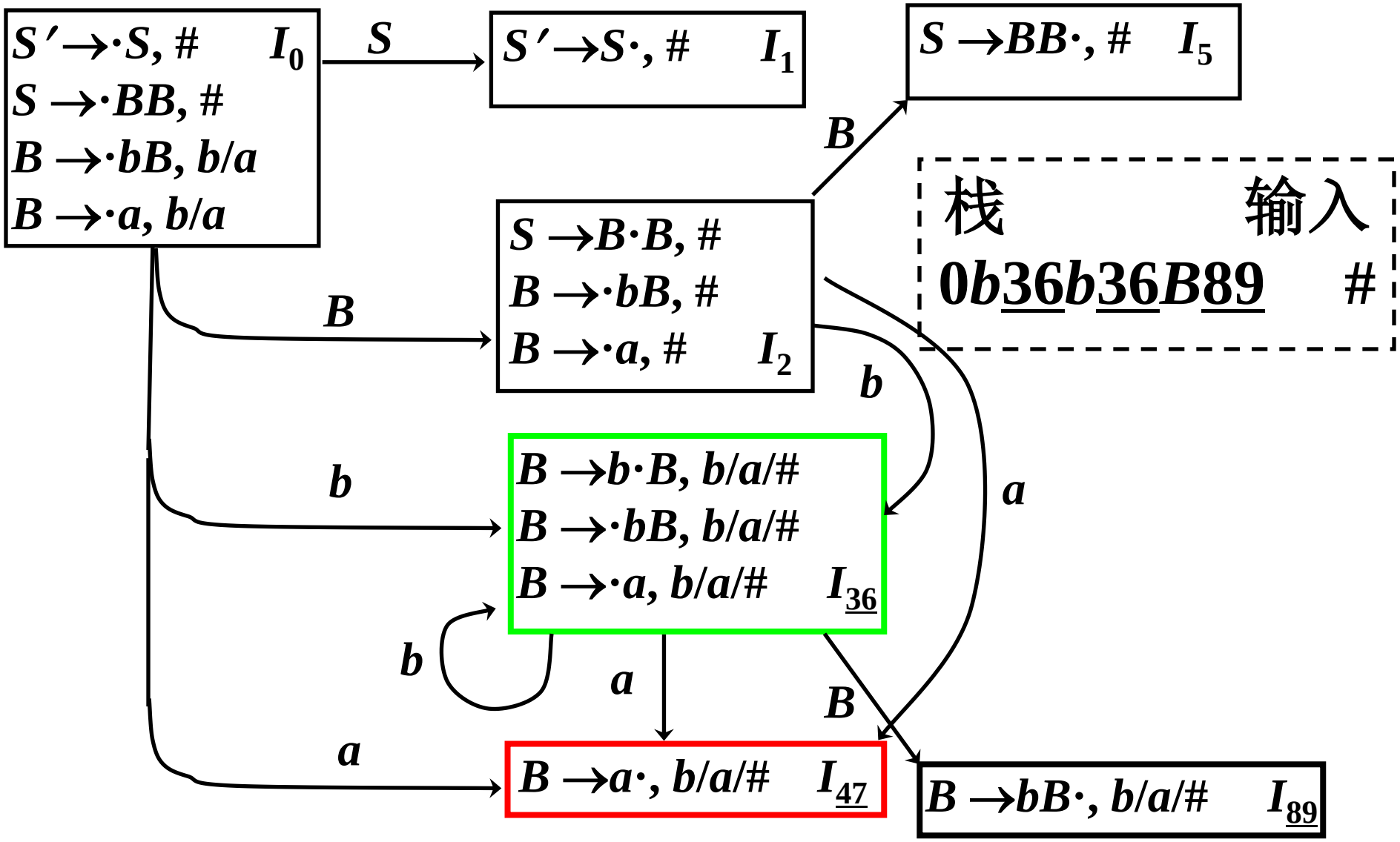
LALR特征(3)


S′→·S, #
S →·BB, #
B →·bB, b/a
B →·a, b/a
I0
S
S′→S·, #
I1
S →BB·, #
I5
B
B
S →B·B, #
B →·bB, #
B →·a, #
I2
b
b
B →b·B, b/a/#
B →·bB, b/a/#
B →·a, b/a/#
I36
a
b
a
B
a
B →a·, b/a/#
I47
B →bB·, b/a/#
I89
栈
输入
0b36b36
a#

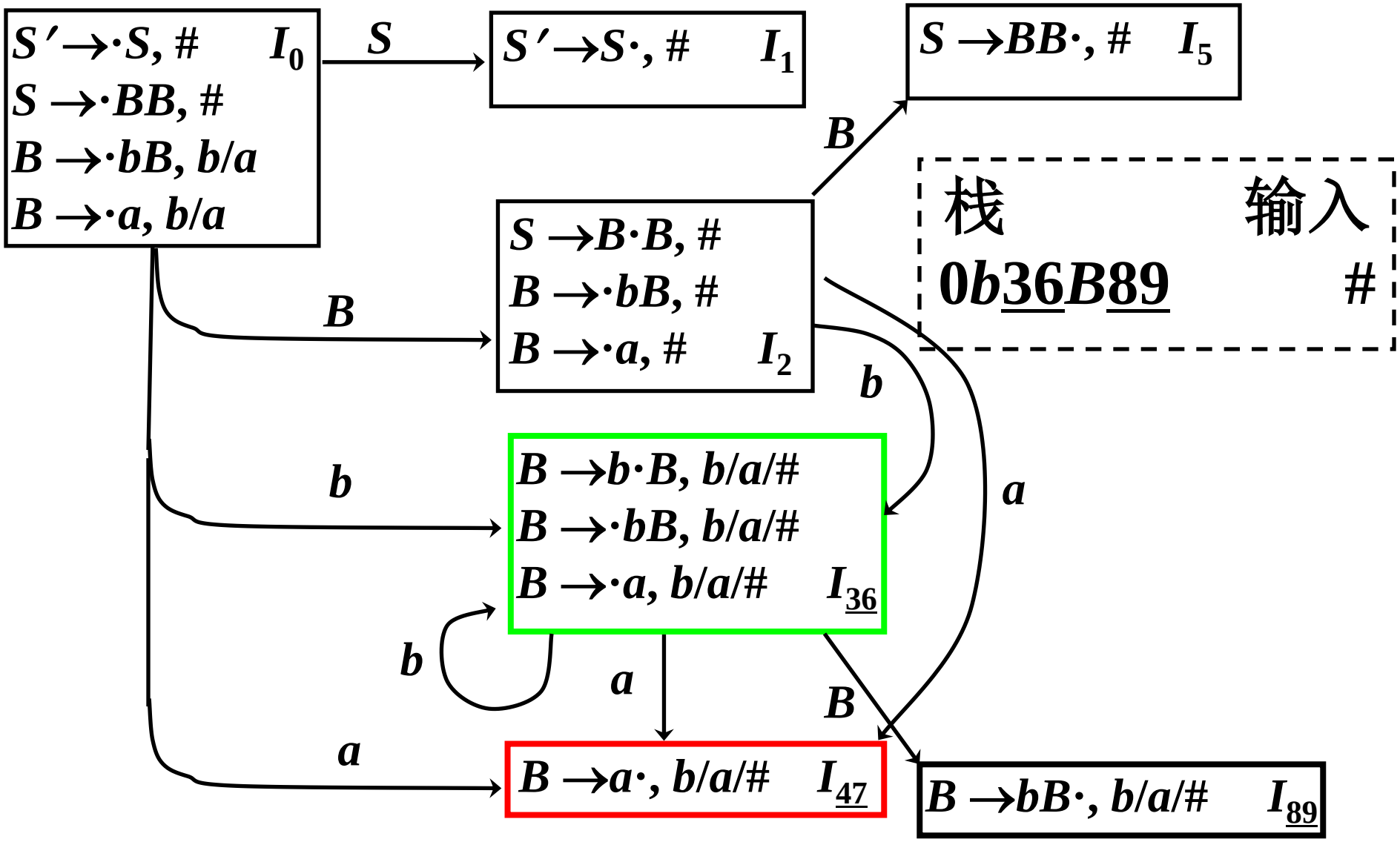
LALR特征(3)


S′→·S, #
S →·BB, #
B →·bB, b/a
B →·a, b/a
I0
S
S′→S·, #
I1
S →BB·, #
I5
B
栈 输入
0b36b36a47 #
S →B·B, #
B →·bB, #
B →·a, #
I2
B
b
B →b·B, b/a/#
B →·bB, b/a/#
B →·a, b/a/#
I36
b
a
b
a
B
a
B →a·, b/a/#
I47
B →bB·, b/a/#
I89

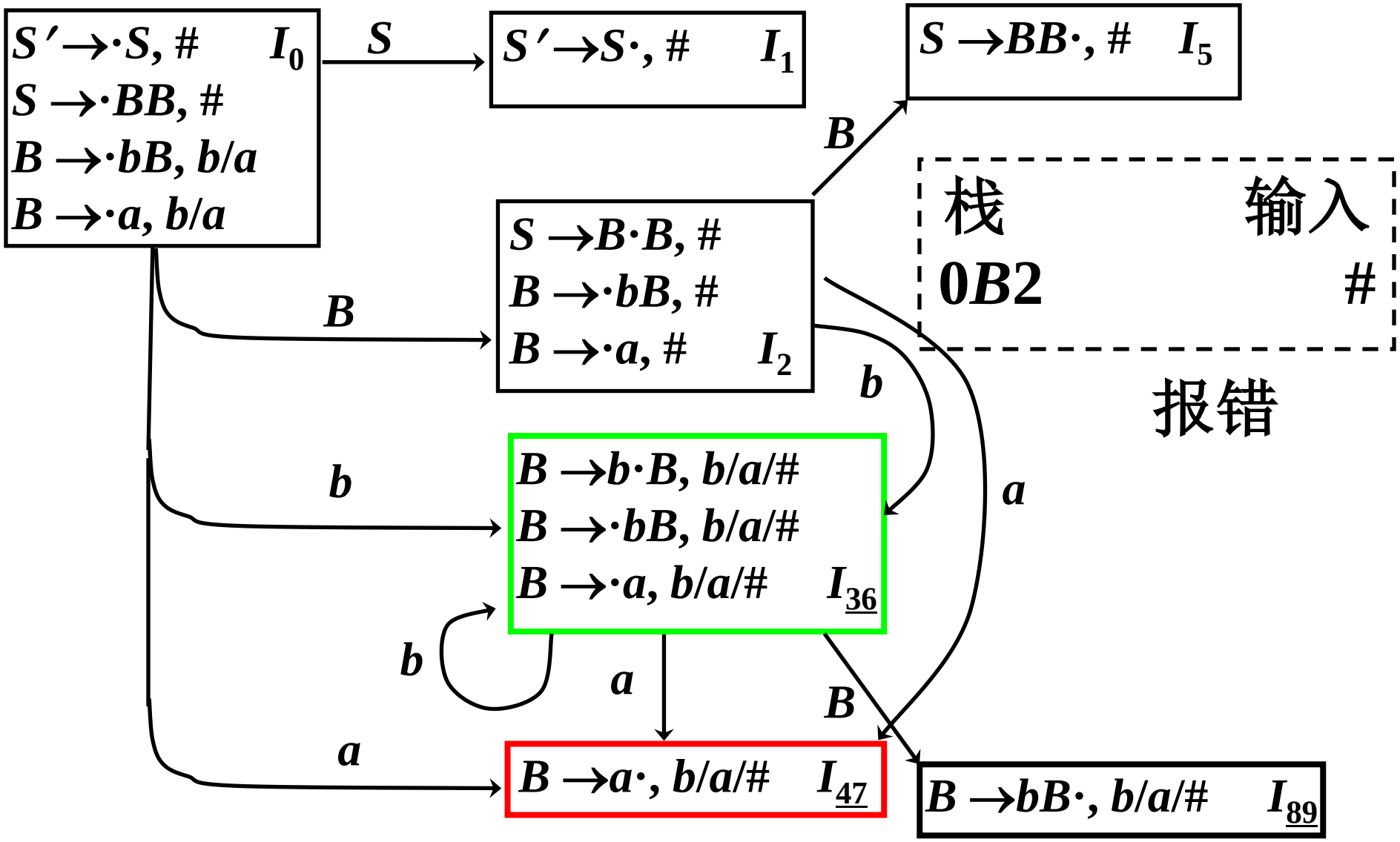
LALR特征(3)


S′→·S, #
S →·BB, #
B →·bB, b/a
B →·a, b/a
I0
S
S′→S·, #
I1
S →BB·, #
I5
B
栈 输入
0b36b36B89 #
S →B·B, #
B →·bB, #
B →·a, #
I2
B
b
B →b·B, b/a/#
B →·bB, b/a/#
B →·a, b/a/#
I36
b
a
b
a
B
a
B →a·, b/a/#
I47
B →bB·, b/a/#
I89

LALR特征(3)


S′→·S, #
S →·BB, #
B →·bB, b/a
B →·a, b/a
I0
S
S′→S·, #
I1
S →BB·, #
I5
B
栈
输入
0b36B89
#
S →B·B, #
B →·bB, #
B →·a, #
I2
B
b
a
B →b·B, b/a/#
B →·bB, b/a/#
B →·a, b/a/#
I36
b
a
B
a
B →a·, b/a/#
I47
B →bB·, b/a/#
I89

LALR特征(3)


S′→·S, #
S →·BB, #
B →·bB, b/a
B →·a, b/a
I0
S
S′→S·, #
I1
S →BB·, #
I5
B
B
S →B·B, #
B →·bB, #
B →·a, #
I2
b
a
栈
输入
0B2
#
报错
b
B →b·B, b/a/#
B →·bB, b/a/#
B →·a, b/a/#
I36
b
a
B
a
B →a·, b/a/#
I47
B →bB·, b/a/#
I89

课后作业

- P134-5

小结

- 自下而上的分析方法
  1. LR(0)方法
  2. SLR(1)方法
  3. 规范LR(1)方法
  4. LALR(1)方法
- LR语法分析程序

LR(0)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“sj”。
2. 若项目$A\rightarrow\alpha\cdot$属于I_k，那么，对任何终结符a(或结束符#)，置ACTION[k,a]为 “rj”(假定产生式$A\rightarrow\alpha$是文法G′的第j个产生式)。
3. 若项目$S'\rightarrow S\cdot$属于I_k，则置ACTION[k,#]为 “acc”。
4. 若$GO(I_k,A) = I_j$，A为非终结符，则置GOTO[k,A]=j。
5. 分析表中凡不能用规则1至4填入信息的空白格均置上“报错标志”。

SLR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 “sj”；

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

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

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

5. 分析表中凡不能用规则1至4填入信息的空白格均置上“出错标志”。

LR(1)分析表的ACTION和GOTO子表构造方法

1. 若项目$A\rightarrow\alpha\cdot a\beta$,b属于$I_k$且$GO(I_k,a)=I_j$，a为终结符，则置ACTION[k,a]为 “sj”；

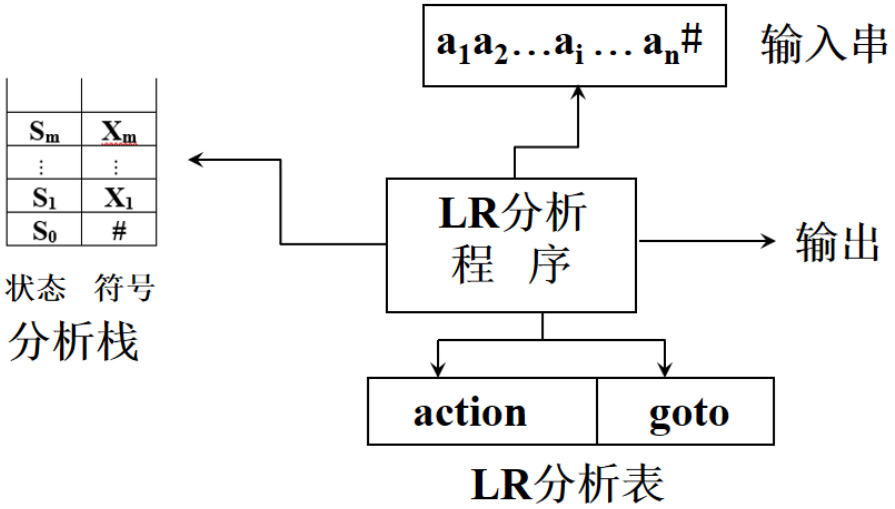
2. 若项目$A\rightarrow\alpha\cdot$,b属于I_k，那么，对任何终结符a，a=b，置ACTION[k,a]为 “rj”；其中，假定$A\rightarrow\alpha$为文法G′的第j个产生式；

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

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

5. 分析表中凡不能用规则1至4填入信息的空白格均置上“出错标志”。

LR语法分析程序




```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

Figure 4.36: LR-parsing program

Book: Compilers Principles Techniques and Tools (2nd Edition) P251

二义文法

二义文法的应用

二义文法的特点：

- 二义文法决不是LR文法
- 简洁、自然
- 可以用文法以外的信息来消除二义
- 语法分析的效率高（基于消除二义后得到的分析表）

举例：二义文法　　$E \to E + E \mid E * E \mid (E) \mid \text{id}$

非二义的文法：

$E \to E + T \mid T$

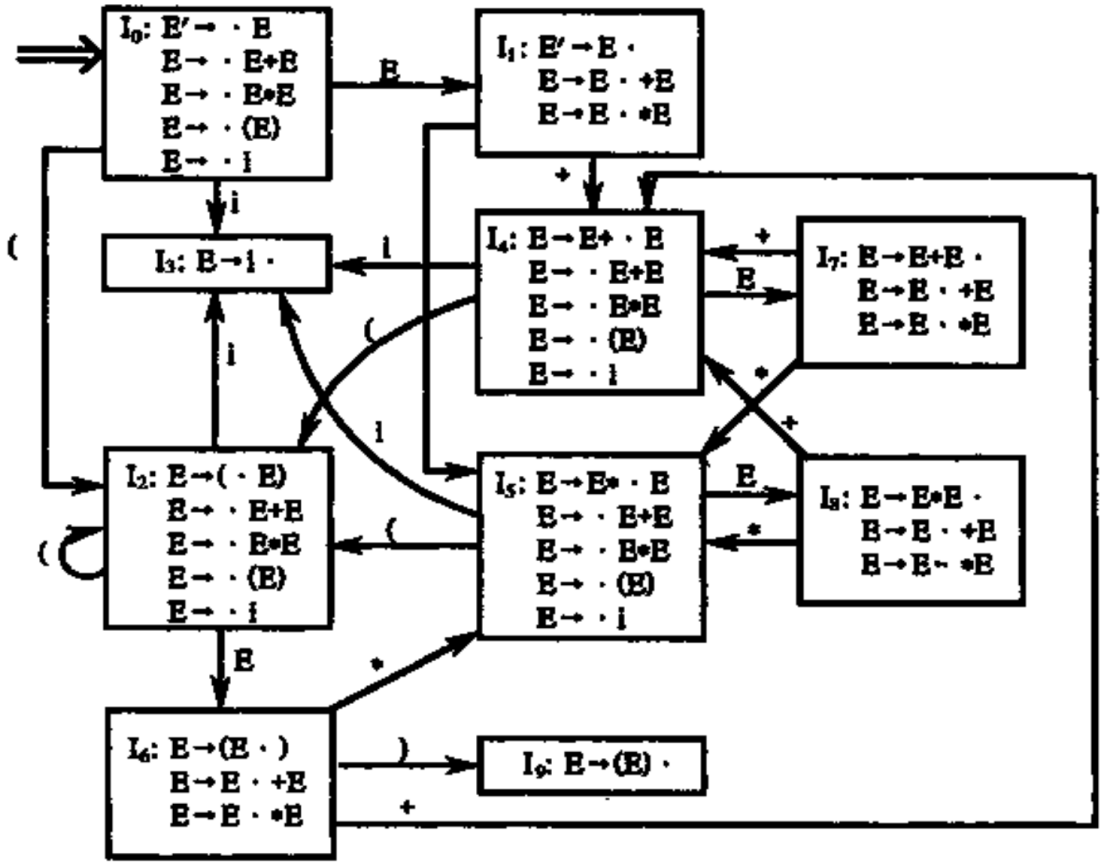
$T \to T * F \mid F$

$F \to (E) \mid \text{id}$

二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \rightarrow E + E \mid E * E \mid (E) \mid \mathrm{id}$



二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

规定： *优先级高于+，两者都是左结合

LR(0)项目集I_7

$$E \to E + E\cdot$$
$$E \to E\cdot + E$$
$$E \to E\cdot * E$$

二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \text{id}$

  规定： *优先级高于+，两者都是左结合

LR(0)项目集I_7（P124图5.11）

$E \to E + E\cdot$

$E \to E\cdot + E$

$E \to E\cdot * E$

id + id + id

面临+，归约

二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

  规定： *优先级高于+，两者都是左结合

LR(0)项目集I_7

$E \to E + E\cdot$

$E \to E\cdot + E$　　id + id　　+ id

$E \to E\cdot * E$　　id + id　　* id

面临+，归约　　面临*，移进

二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

  规定： *优先级高于+，两者都是左结合

LR(0)项目集I_7

$E \to E + E\cdot$

$E \to E\cdot + E$　　id + id　　+ id

$E \to E\cdot * E$　　id + id　　* id

面临+，归约　　面临*，移进

二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \text{id}$

规定： *优先级高于+，两者都是左结合

LR(0)项目集I_8

$E \to E * E \cdot$
$E \to E \cdot + E$
$E \to E \cdot * E$

二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

  规定： *优先级高于+，两者都是左结合

LR(0)项目集I_8

$E \to E * E \cdot$

$E \to E \cdot + E$

$E \to E \cdot * E$

id * id + id

面临+，归约

二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

  规定： *优先级高于+，两者都是左结合

LR(0)项目集I_8

$E \to E * E \cdot$

$E \to E \cdot + E$　　id * id　　+ id

$E \to E \cdot * E$　　id * id　　* id

面临+，归约　　面临*，归约

二义文法的应用

1. 使用文法以外信息来解决分析动作冲突

- 例 二义文法 $E \to E + E \mid E * E \mid (E) \mid \mathrm{id}$

  规定： *优先级高于+，两者都是左结合

LR(0)项目集I_8

$E \to E * E \cdot$

$E \to E \cdot + E$　　id * id　　+ id

$E \to E \cdot * E$　　id * id　　* id

面临+，归约　　面临*，归约

自我检测

- 思考下面二义文法如何解决LR冲突问题？
  - S → iSeS
  - S → iS
  - S → a

  (提示：可以规定else搭配最近的一个if…then)

本 章 要 点

- 自下而上的分析方法
  - 自下而上分析基本问题
  - 规范规约
  - 算符优先分析方法
  - LR分析法
    - LR(0)方法
    - SLR(1)方法
    - 规范LR(1)方法
    - LALR(1)方法
    - LR方法如何用于二义文法

Dank u
Dutch

Merci
French

Спасибо
Russian

Gracias
Spanish

شكراً
Arabic

감사합니다
Korean

Tack så mycket
Swedish

धन्यवाद
Hindi

תודה רבה
Hebrew

Obrigado
Brazilian Portuguese

Dankon
Esperanto

Thank You !

谢谢
Chinese

ありがとうございます
Japanese

Trugarez
Breton

Danke
German

Tak
Danish

Grazie
Italian

நன்றி
Tamil

děkuji
Czech

ขอบคุณ
Thai

go raibh maith agat
Gaelic