



编译原理

第五章 语法分析 自下而上分析

所谓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；或者说从语法树的末端开始，步步向上“归约”，直到根结。

$$G(E): E \rightarrow i \mid E+E \mid E-E \mid E^*E \mid E/E \mid$$

(E)

i*i+i

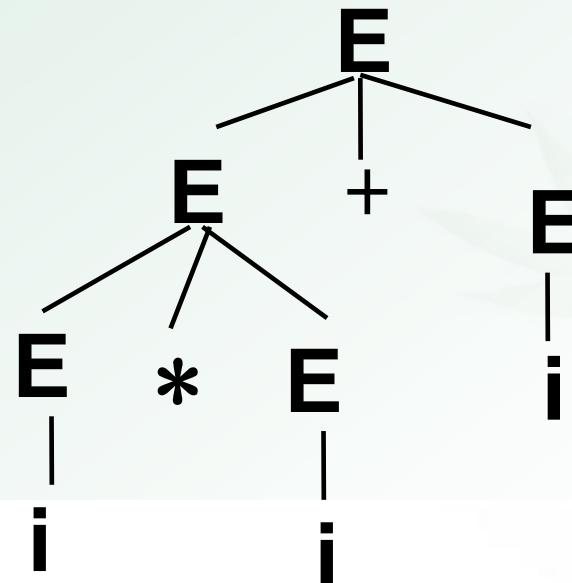
E*i+i

E^*E+i

E+i

E+E

E



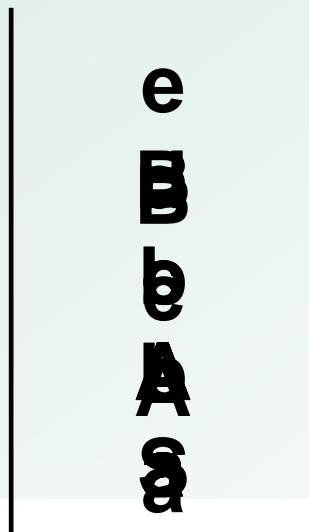
5.1.1 归约

- 采用“移进—归约”思想进行自下而上分析。
- 基本思想：用一个寄存符号的先进后出栈，把输入符号一个一个地移进到栈里，当栈顶形成某个产生式的候选式时，即把栈顶的这一部分替换成(归约为)该产生式的左部符号。

• 例：设文法G(S)：

- (1) $S \rightarrow aAcBe$
- (2) $A \rightarrow b$
- (3) $A \rightarrow Ab$
- (4) $B \rightarrow d$

试对abbcde进行“移进—归约”分析。



abbcde

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

分析树和语法树不一定一致。

自下而上分析过程：边输入单词符号，边归约。

核心问题：识别可归约串

自下而上分析的关键问题： 如何确定可归约串？

通过自底向上分析算法中的优先关系来计算

- 简单优先分析法（规范规约）：寻找句柄
- 算符优先分析法：寻找最左素短语

一、自底向上优先分析法概述

优先分析法可分简单优先分析法和算符优先分析法。

①**简单优先分析法**：按一定原则求出文法中所有符号（终结符和非终结符）的优先关系，按这种关系求出句柄。（规范归约——从左向右的规约）；

②**算符优先分析法**：只考虑算符（终结符）之间的优先关系，不考虑非终结符之间的优先关系。按这种关系求出最左素短语。（不规范归约）

简单优先分析法：准确规范，但分析效率低，实际使用价值不大；

算符优先分析法：不规范，但分析速度快，适于实际使用。

5.1.2 规范归约

- 定义：令G是一个文法，S是文法的开始符号，假定 $\alpha\beta\delta$ 是文法G的一个句型，如果有

$$S \xrightarrow{*} \alpha A \delta \quad \text{且} \quad A \xrightarrow{+} \beta$$

则 β 称是句型 $\alpha\beta\delta$ 相对于非终结符A的短语。

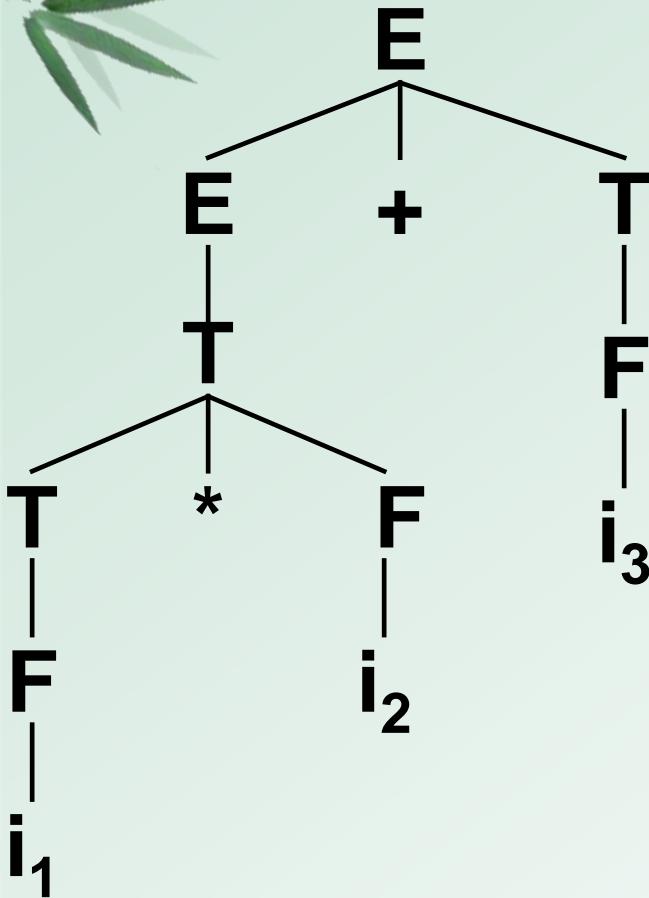
特别是，如果有 $A \Rightarrow \beta$ ，则称 β 是句型 $\alpha\beta\delta$ 相对于规则 $A \rightarrow \beta$ 的直接短语。一个句型的最左直接短语称为该句型的句柄。

考慮文法 $G(E)$: $E \rightarrow T \mid E+T$
 $T \rightarrow F \mid T*F$
 $F \rightarrow (E) \mid i$

和句型 $i_1*i_2+i_3$:

$$\begin{aligned} E &\Rightarrow E+T \Rightarrow E+F \Rightarrow E+i_3 \Rightarrow T+i_3 \\ &\Rightarrow T*F+i_3 \Rightarrow T*i_2+i_3 \Rightarrow F*i_2+i_3 \\ &\Rightarrow i_1*i_2+i_3 \end{aligned}$$

- 短语: $i_1, i_2, i_3, i_1*i_2, i_1*i_2+i_3$
- 直接短语: i_1, i_2, i_3
- 句柄: i_1



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

- 可用句柄来对句子进行归约

句型 归约规则

a b c d e (2) $A \rightarrow b$

a A b c d e (3) $A \rightarrow Ab$

a A c d e (4) $B \rightarrow d$

a A c B e (1) $S \rightarrow a A c B e$

S

- 定义：假定 α 是文法G的一个句子，我们称序列

$\alpha_n, \alpha_{n-1}, \dots, \alpha_0$

是的一个规范归约，如果此序列满足：

1 $\alpha_n = \alpha$

2 α_0 为文法的开始符号，即 $\alpha_0 = S$

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

把上例倒过来写，则得到：

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

显然这是一个最右推导。

规范归约是关于一个最右推导的逆过程

最左归约

规范推导

由规范推导推出的句型称为规范句型。

5.1.3 符号栈的使用和分析树的表示

- 栈是语法分析的一种基本数据结构。' #' 作为栈底符号
- 考虑文法G(E)：

$$\begin{aligned} E &\rightarrow T \mid E+T \\ T &\rightarrow F \mid T*F \\ F &\rightarrow (E) \mid i \end{aligned}$$

输入串为 $i_1 * i_2 + i_3$ ，分析步骤为：

- $G(E)$:

$$E \rightarrow T \mid E+T$$

$$T \rightarrow F \mid T^*F$$

$$F \rightarrow (E) \mid i$$

<u>步骤</u>	<u>符号栈</u>	<u>输入串</u>	<u>动作</u>
0	#	i1*i2+i3#	预备
1	#i1	*i2+i3#	进
2	#F	*i2+i3#	归, 用 $F \rightarrow i$
3	#T	*i2+i3#	归, 用 $T \rightarrow F$
4	#T*	i2+i3#	进

- $G(E)$:

$$E \rightarrow T \mid E+T$$

$$T \rightarrow F \mid T^*F$$

$$F \rightarrow (E) \mid i$$

<u>步骤</u>	<u>符号栈</u>	<u>输入串</u>	<u>动作</u>
4	#T*	i 2+i 3#	进
5	#T*i 2	+i 3#	进
6	#T*F	+i 3#	归, 用 $F \rightarrow i$
7	#T	+i 3#	归, 用 $T \rightarrow T^*F$
8	#E	+i 3#	归, 用 $E \rightarrow T$
9	#E+	i 3#	进

- $G(E)$:

$$E \rightarrow T \mid E+T$$

$$T \rightarrow F \mid T^*F$$

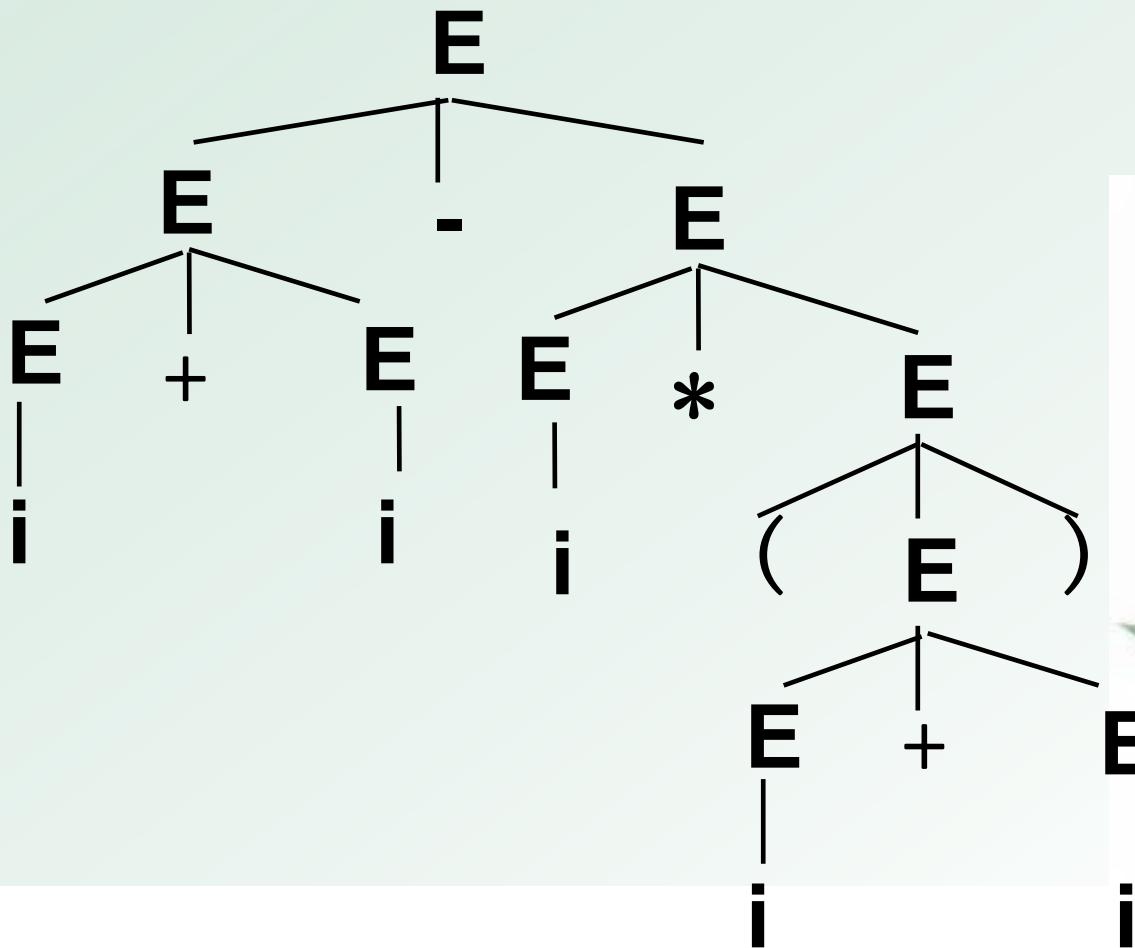
$$F \rightarrow (E) \mid i$$

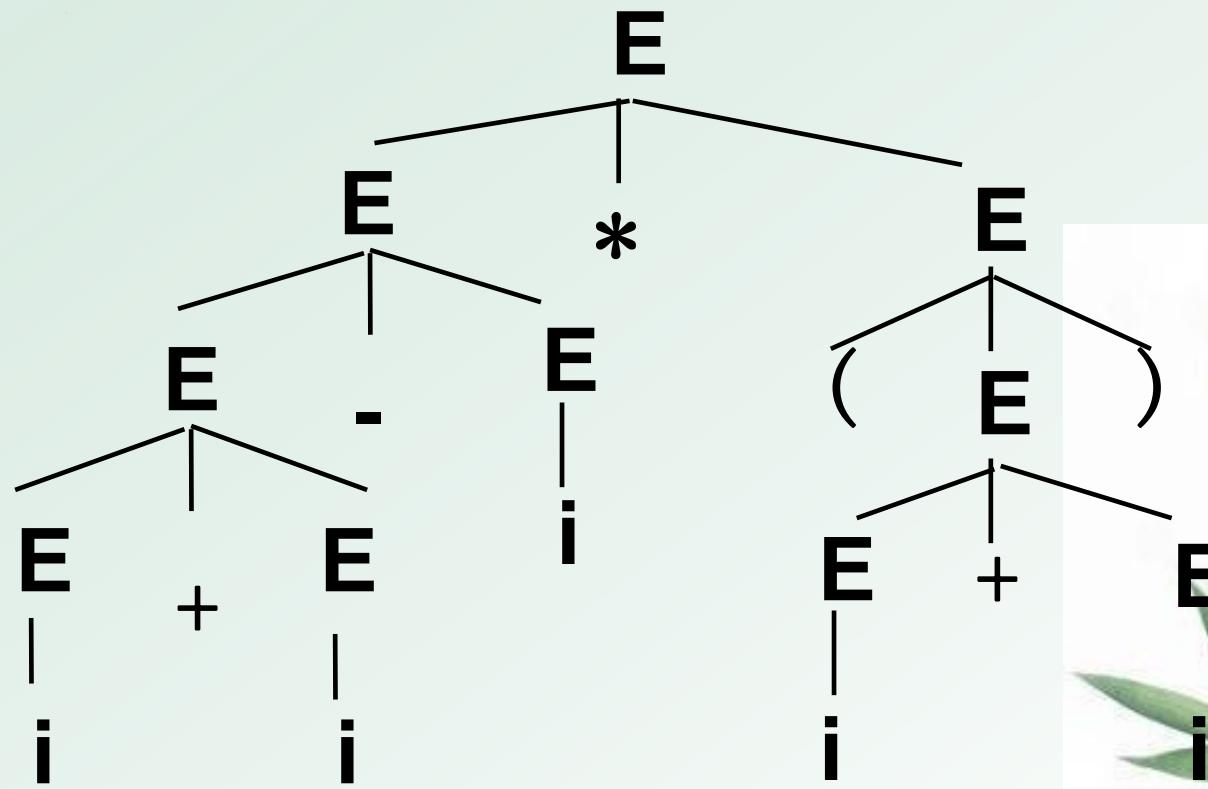
<u>步骤</u>	<u>符号栈</u>	<u>输入串</u>	<u>动作</u>
9	#E+	i3#	进
10	#E+i3	#	进
11	#E+F	#	归, 用 $F \rightarrow i$
12	#E+T	#	归, 用 $T \rightarrow F$
13	#E	#	归, 用 $E \rightarrow E+T$
14	#E	#	接受

5. 2 算符优先分析

- 四则运算的优先规则：
先乘除后加减，同级从左到右
- 考虑二义文法文法G(E)：
$$G(E) : E \rightarrow i \mid E+E \mid E-E \mid E*E \mid E/E \mid (E)$$
- 它的句子有几种不同的规范规约。
- 归约即计算表达式的值。归约顺序不同，则计算的顺序也不同，结果也不一样。
- 如果规定算符的优先次序，并按这种规定进行归约，则归约过程是唯一的。

例如：句子 $i+i-i*(i+i)$





句子 $i+i-i*(i+i)$ 的归约过程是：

- (1) $i+i-i*(i+i)$
- (2) $E+i-i*(i+i)$
- (3) $E+E-i*(i+i)$
- (4) $E-i*(i+i)$
- (5) $E-E*(i+i)$
- (6) $E-E*(E+i)$
- (7) $E-E*(E+E)$
- (8) $E-E*(E)$
- (9) $E-E*E$
- (10) $E-E$
- (11) E

- 起决定作用的是相邻的两个算符之间的优先关系。
- 所谓**算符优先分析法**就是定义算符之间的某种优先关系，借助于这种关系寻找“**可归约串**”和进行归约。

- 首先必须定义任何两个可能相继出现的终结符a与b的优先关系

三种关系

$a < \cdot b$ a的优先级低于b

$a = \cdot b$ a的优先级等于b

$a \cdot > b$ a的优先级高于b

- 注意：与数学上的 $<>=$ 不同



5.2.1 算符优先文法及优先表构造

- 一个文法，如果它的任一产生式的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：

…QR…

则我们称该文法为**算符文法**。

- 约定：
 - a、b代表任意终结符；
 - P、Q、R代表任意非终结符；
 - ‘…’ 代表由终结符和非终结符组成的任意序列，包括空字。

- 假定G是一个不含 ε -产生式的算符文法。对于任何一对终结符a、b，我们说：
 - $a = \cdot b$ 当且仅当文法G中含有形如 $P \rightarrow \cdots ab \cdots$ 或 $P \rightarrow \cdots aQb \cdots$ 的产生式；
 - $a < \cdot b$ 当且仅当G中含有形如 $P \rightarrow \cdots aR \cdots$ 的产生式，而 $R \xrightarrow{+} b \cdots$ 或 $R \xrightarrow{+} Qb \cdots$ ；
 - $a \cdot > b$ 当且仅当G中含有形如 $P \rightarrow \cdots Rb \cdots$ 的产生式，而 $R \xrightarrow{+} \cdots a$ 或 $R \xrightarrow{+} \cdots aQ$ 。

- 如果一个算符文法G中的任何终结符对(a, b)至多只满足下述三关系之一：

$$a = \cdot b, \quad a < \cdot b, \quad a \cdot > b$$

则称G是一个算符优先文法。

- 例：考慮下面的文法 $G(E)$ ：

$$\begin{array}{ll}
 (1) & E \rightarrow E + T \quad | \quad T \\
 (2) & T \rightarrow T * F \quad | \quad F \\
 (3) & F \rightarrow P \uparrow F \quad | \quad P \\
 (4) & P \rightarrow (E) \quad | \quad i
 \end{array}$$

- 由第(4)条规则，有 ‘(’ = · ‘)’ ；
- 由规则 $E \rightarrow E + T$ 和 $T \Rightarrow T * F$ ，有 + < · * ；
- 由(2) $T \rightarrow T * F$ 和 (3) $F \rightarrow P \uparrow F$ ，可得 * < · ↑ ；
- 由(1) $E \rightarrow E + T$ 和 $E \Rightarrow E + T$ ，可得 + · > + ；
- 由(3) $F \rightarrow P \uparrow F$ 和 $F \Rightarrow P \uparrow F$ ，可得 ↑ < · ↑ 。
- 由(4) $P \rightarrow (E)$ 和 $E \Rightarrow E + T \Rightarrow T + T \Rightarrow T * F + T \Rightarrow F * F + T$
 $\Rightarrow P \uparrow F * F + T \Rightarrow i \uparrow F * F + T$
 有 (< · +、(< · *、(< · ↑ 和 (< · i)。

- 从算符优先文法G构造优先关系表的算法。
- 通过检查G的每个产生式的每个候选式，可找出所有满足 $a = \cdot b$ 的终结符对。

1. $a = \cdot b$ 当且仅当文法G中含有形如 $P \rightarrow \cdots ab \cdots$ 或 $P \rightarrow \cdots aQb \cdots$ 的产生式；

- 确定满足关系 $a < \cdot b$ 和 $a \cdot > b$ 的所有终结符对：

2. $a < \cdot b$ 当且仅当G中含有形如 $P \rightarrow \cdots aR \cdots$ 的产生式，而 $R \xrightarrow{+} b \cdots$ 或 $R \xrightarrow{+} Qb \cdots$ ；

3. $a \cdot > b$ 当且仅当G中含有形如 $P \rightarrow \cdots Rb \cdots$ 的产生式，而 $R \xrightarrow{+} \cdots a$ 或 $R \xrightarrow{+} \cdots aQ$ 。

- 从算符优先文法G构造优先关系表的算法。
- 通过检查G的每个产生式的每个候选式，可找出所有满足 $a = \cdot b$ 的终结符对。
- 确定满足关系 $\langle \cdot \cdot \rangle$ 的所有终结符对：
 - 首先需要对G的每个非终结符P构造两个集合FIRSTVT(P)和LASTVT(P)。

$$FIRSTVT(P) = \{a \mid P \xrightarrow{+} a \cdots, \text{或 } P \xrightarrow{+} Q a \cdots, a \in V_T \text{ 而 } Q \in V_N\}$$

$a < \cdot \cdot b$ 当且仅当G中含有形如 $P \rightarrow \cdots a R \cdots$ 的产生式，而 $R \xrightarrow{+} b \cdots$ 或 $R \xrightarrow{+} Q b \cdots$ ；

- 从算符优先文法G构造优先关系表的算法。
- 通过检查G的每个产生式的每个候选式，可找出所有满足 $a = \cdot b$ 的终结符对。
- 确定满足关系 $\langle \cdot \cdot \rangle$ 的所有终结符对：
 - 首先需要对G的每个非终结符P构造两个集合FIRSTVT(P)和LASTVT(P)：

$$LASTVT(P) = \{a \mid P \xrightarrow{+} \cdots a, \text{或} P \xrightarrow{+} \cdots aQ, a \in V_T \text{ 而 } Q \in V_N\}$$

3. $a \cdot \cdot > b$ 当且仅当G中含有形如 $P \rightarrow \cdots Rb \cdots$ 的产生式，而 $R \xrightarrow{+} \cdots a$ 或 $R \xrightarrow{+} \cdots aQ$ 。

- 从算符优先文法G构造优先关系表的算法。
- 通过检查G的每个产生式的每个候选式，可找出所有满足 $a = \cdot b$ 的终结符对。
- 确定满足关系 $\langle \cdot \text{ 和 } \cdot \rangle$ 的所有终结符对：
 - 首先需要对G的每个非终结符P构造两个集合FIRSTVT(P)和LASTVT(P)：

$$FIRSTVT(P) = \{a \mid P \xrightarrow{+} a \dots, \text{或} P \xrightarrow{+} Q a \dots, a \in V_T \text{ 而 } Q \in V_N\}$$

比較 $FIRST(\alpha) = \{a \mid \alpha \Rightarrow a \dots, a \in V_T\}$

$$LASTVT(P) = \{a \mid P \xrightarrow{+} \dots a, \text{或} P \xrightarrow{+} \dots a Q, a \in V_T \text{ 而 } Q \in V_N\}$$

比較 $FOLLOW(A) = \{a \mid S \xrightarrow{*} \dots A a \dots, a \in V_T\}$

□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 \prec 和 \succ 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 $b \in FIRSTVT(P)$ ，有 $a \prec \cdot b$ 。

➤假定有个产生式的一个候选形为

...Pb...

那么，对任何 $a \in LASTVT(P)$ ，有 $a \cdot \succ b$ 。

$$FIRSTVT(P) = \{a \mid P \xrightarrow{+} a \cdots, \text{或 } P \xrightarrow{+} Qa \cdots, a \in V_T \text{ 而 } Q \in V_N\}$$

- 首先讨论构造集合FIRSTVT (P) 的算法。
- 按其定义，可用下面两条规则来构造集合FIRSTVT (P)：
 1. 若有产生式 $P \rightarrow a \cdots$ 或 $P \rightarrow Qa \cdots$ ，则 $a \in FIRSTVT (P)$ ；
 2. 若 $a \in FIRSTVT (Q)$ ，且有产生式 $P \rightarrow Q \cdots$ ，则 $a \in FIRSTVT (P)$ 。

- 数据结构：
 - 布尔数组 $F[P, a]$ ，使得 $F[P, a]$ 为真的条件是，当且仅当 $a \in FIRSTVT(P)$ 。开始时，按上述的规则(1)对每个数组元素 $F[P, a]$ 赋初值。
 - 栈 $STACK$ ，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。

- 运算：

- 如果栈STACK不空，就将顶项逐出，记此项为(Q, a)。对于每个形如

$P \rightarrow Q \dots$

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。

- 如果把这个算法稍为形式化一点，我们可得如下所示的一个程序(包括一个过程和主程序)：

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

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 $P \rightarrow a \dots$ 或 $P \rightarrow Qa \dots$ 的产生式 DO

 INSERT(P, a);

WHILE STACK 非空 DO

BEGIN

 把STACK的顶项，记为(Q, a)，上托出去；

 FOR 每条形如 $P \rightarrow Q \dots$ 的产生式 DO

 INSERT(P, a);

 END OF WHILE;

END

- 这个算法的工作结果得到一个二维数组F，从它可得任何非终结符P的FIRSTVT。

$$\text{FIRSTVT}(P) = \{a \mid F[P, a] = \text{TRUE}\}$$

- 同理，可构造计算LASTVT的算法。



$$LASTVT(P) = \{a \mid P \xrightarrow{+} \cdots a, \text{或} P \xrightarrow{+} \cdots aQ, a \in V_T \text{ 而 } Q \in V_N\}$$

- 构造集合LASTVT (P) 的算法。
- 按其定义，可用下面两条规则来构造集合 LASTVT (P)：
 1. 若有产生式 $P \rightarrow \cdots a$ 或 $P \rightarrow \cdots aQ$ ，则 $a \in LASTVT (P)$ ；
 2. 若 $a \in LASTVT (Q)$ ，且有产生式 $P \rightarrow \cdots Q$ ，则 $a \in LASTVT (P)$ 。

- 使用每个非终结符P的FIRSTVT (P) 和LASTVT (P) ,就能够构造文法G的优先表。构造优先表的算法是：



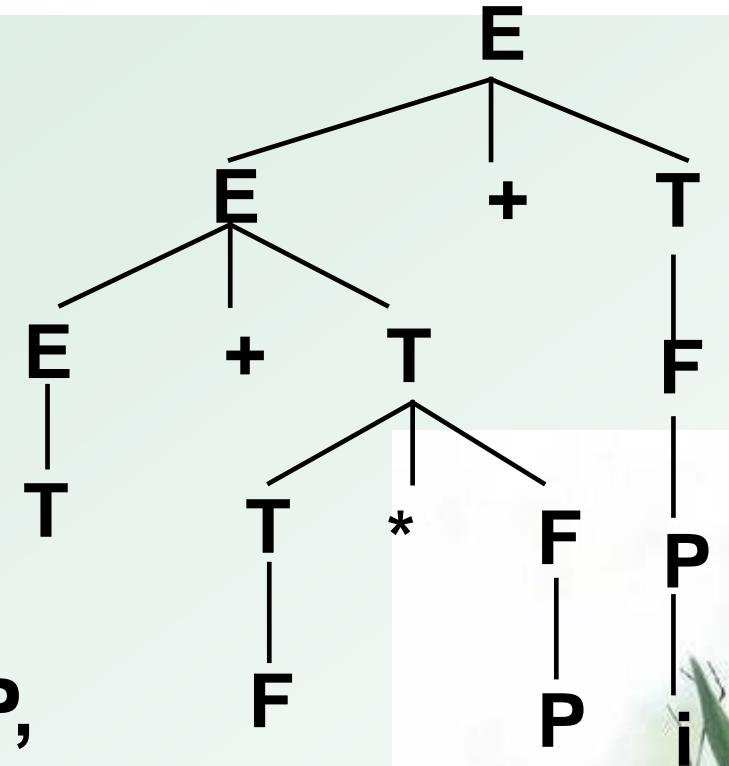
```
FOR 每条产生式P→X1X2…Xn DO  
  FOR i := 1 TO n-1 DO  
    BEGIN  
      IF Xi和Xi+1均为终结符 THEN 置Xi Xi+1  
      IF i ≤ n-2且Xi和Xi+2都为终结符  
        但Xi+1为非终结符 THEN 置Xi Xi+2;  
      IF Xi为终结符而Xi+1为非终结符 THEN  
        FOR FIRSTVT(Xi+1) 中的每个a DO  
          置 Xi a;  
      IF Xi为非终结符而Xi+1为终结符 THEN  
        FOR LASTVT(Xi) 中的每个a DO  
          置 a Xi+1  
    END
```

5.2.2 算符优先分析算法

- 可归约串，句型，短语，直接短语，句柄，规范归约。
- 一个文法G的句型的**素短语**是指这样一个短语，它至少含有一个终结符，并且，除它自身之外不再含任何更小的素短语。
- **最左素短语**是指处于句型最左边的那个素短语。

- 考慮下面的文法 $G(E)$:

$$\begin{array}{l}
 (1) \quad E \rightarrow E + T \quad | \quad T \\
 (2) \quad T \rightarrow T * F \quad | \quad F \\
 (3) \quad F \rightarrow P \quad | \quad F \quad | \quad P \\
 (4) \quad P \rightarrow (E) \quad | \quad i
 \end{array}$$



句型: $T+F*i$

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

- 算符优先文法句型(括在两个#之间)的一般形式写成:

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

其中，每个 a_i 都是终结符， N_i 是可有可无的非终结符。

- 定理：一个算符优先文法G的任何句型的最左素短语是满足如下条件的最左子串 $N_ja_j\cdots N_ia_iN_{i+1}$ ，

$$\begin{array}{cc} a_{j-1} & a_j \\ a_j & a_{j+1}, \end{array}, \dots, \begin{array}{cc} a_{i-1} & a_i \\ a_i & a_{i+1} \end{array}$$

- 算符优先分析算法
- 使用一个符号栈S，用它寄存终结符和非终结符，k代表符号栈S的使用深度。

```
k:=1;  
S[k]:='#';  
REPEAT
```

把下一个输入符号读进a中；

```
IF S[k] ∈ VT THEN j:=  
WHILE S[j] = a DO  
BEGIN
```

REPEAT

Q:=S[j];

IF S[j-1] ∈ V_T THEN j:=j-1 ELSE j:=j-2

UNTIL S[j] = Q;

把S[j+1]...S[k]归约为某个N；

k:=j+1;

S[k]:=N

END OF WHILE;

IF S[j] = a OR S[j] ≠ a THEN

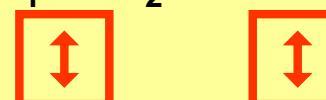
BEGIN k:=k+1; S[k]:=a END

ELSE ERROR /*调用出错诊察程序*/

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。

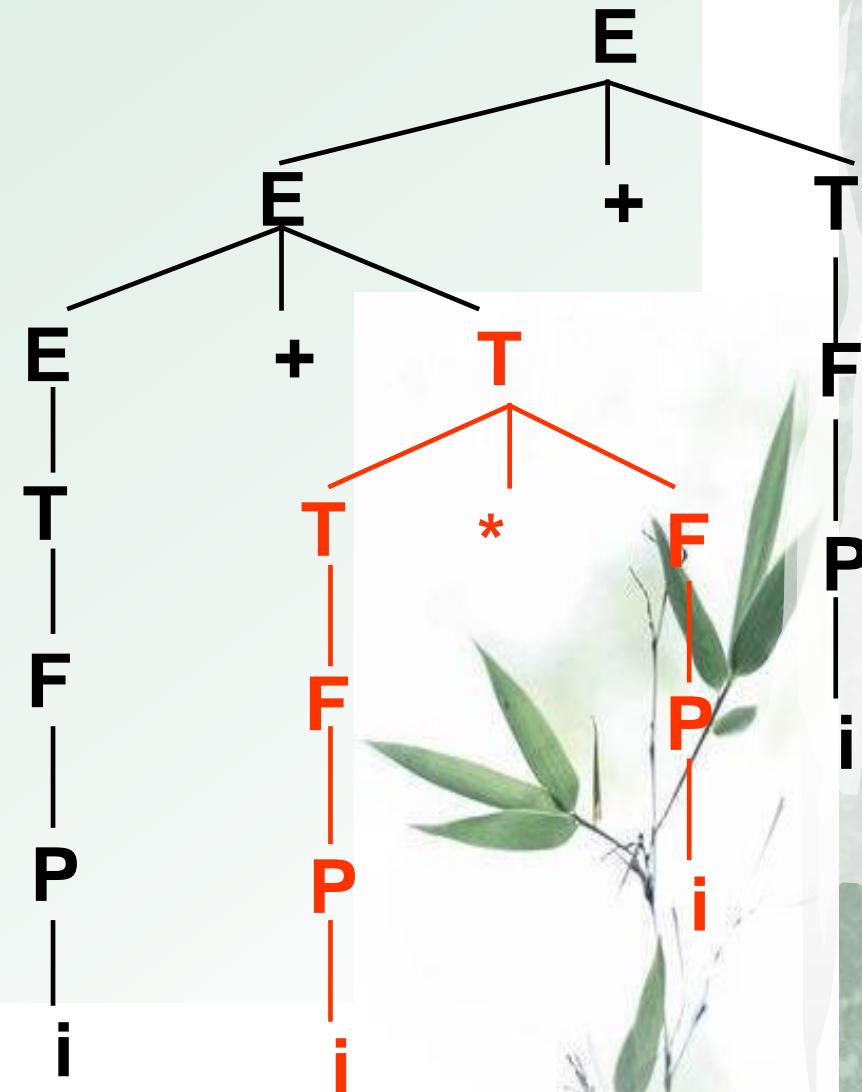
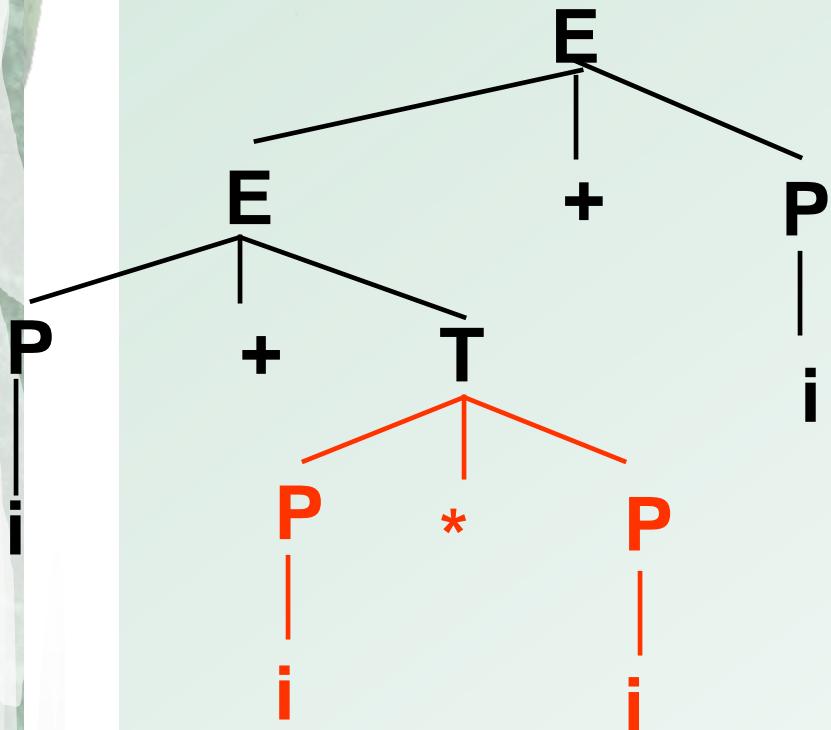
N → X₁ X₂ ... X_{k-j}



S[j+1] S[j+2] ... S[k]

- 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 。

- 算符优先分析一般并不等价于规范归约。



- 考虑下面的文法 $G(E)$:
 - (1) $E \rightarrow E+T \mid T$
 - (2) $T \rightarrow T^*F \mid F$
 - (3) $F \rightarrow P \uparrow F \mid P$
 - (4) $P \rightarrow (E) \mid i$
 的句子 $i+i^*i+i$

- 算符优先分析法特点：
 - 优点：简单，快速
 - 缺点：可能错误接受非法句子，能力有限.
- 算符优先分析法是一种广为应用、行之有效的方法。
 - 用于分析各类表达式
 - ALGOL 60

5. 2. 3 优先函数

- 把每个终结符 θ 与两个自然数 $f(\theta)$ 与 $g(\theta)$ 相对应，使得

若 $\theta_1 < \theta_2$, 则 $f(\theta_1) < g(\theta_2)$

若 $\theta_1 = \theta_2$, 则 $f(\theta_1) = g(\theta_2)$

若 $\theta_1 > \theta_2$, 则 $f(\theta_1) > g(\theta_2)$

f 称为入栈优先函数, g 称为比较优先函数。

- 优点: 便于比较, 节省空间;
- 缺点: 原来不存在优先关系的两个终结符, 由于自然数相对应, 变成可以比较的。要进行一些特殊的判断。

- 文法G(E)

$$(1) \quad E \rightarrow E + T \quad | \quad T$$

$$(2) \quad T \rightarrow T * F \quad | \quad F$$

$$(3) \quad F \rightarrow P \uparrow F \quad | \quad P$$

$$(4) \quad P \rightarrow (E) \quad | \quad i$$

的优先函数如下表

	+	*	\uparrow	()	i	#
F	2	4	4	0	6	6	0
G	1	3	5	5	0	5	0

- 有许多优先关系表不存在优先函数，如：

	a	b
a	= -	- >
b	= -	= -

不存在对应的优先函数f和g

假定存在f和g，则有

$$f(a) = g(a), \quad f(a) > g(b),$$

$$f(b) = g(a), \quad f(b) = g(b)$$

导致如下矛盾：

$$f(a) > g(b) = f(b) = g(a) = f(a)$$

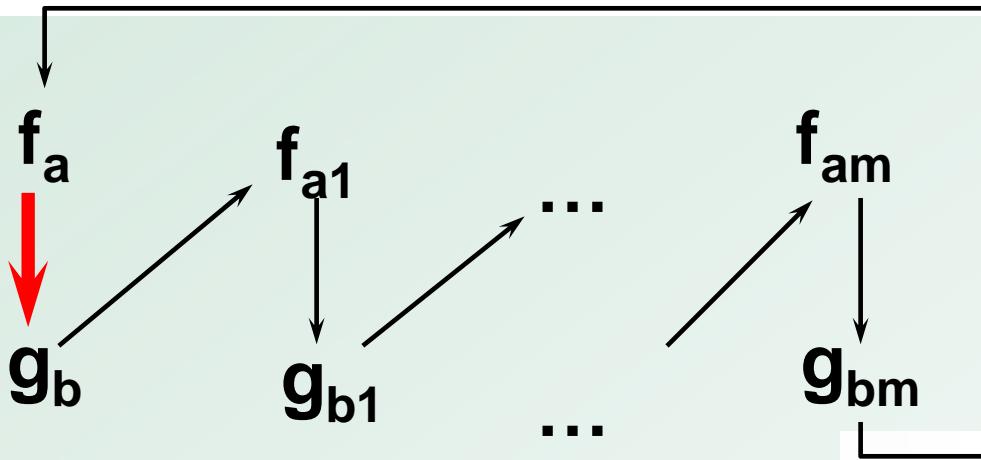
如果优先函数存在，则不唯一（无穷多）

- 如果优先函数存在，则可以通过以下三个步骤从优先表构造优先函数：
 - 1 对于每个终结符 a ，令其对应两个符号 f_a 和 g_a ，画一以所有符号和为结点的方向图。如果 $a \cdot > = \cdot b$ ，则从 f_a 画一条弧至 g_b ，如果 $a \cdot < = \cdot b$ ，则画一条弧从 g_b 至 f_a 。
 - 2 对每个结点都赋予一个数，此数等于从该结点出发所能到达的结点（包括出发点自身）。赋给 f_a 的数作为 $f(a)$ ，赋给 g_a 的数作为 $g(a)$ 。
 - 3 检查所构造出来的函数 f 和 g 是否与原来的关系矛盾。若没有矛盾，则 f 和 g 就是要求的优先函数，若有矛盾，则不存在优先函数。

1 对于每个终结符 a , 令其对应两个符号 f_a 和 g_a , 画一以所有符号和为结点的方向图。如果 $a \cdot > b$, 则从 f_a 画一条弧至 g_b , 如果 $a \cdot < b$, 则画一条弧从 g_b 至 f_a 。

- 现在必须证明: 若 $a \cdot = b$, 则 $f(a) = g(b)$; 若 $a \cdot < b$, 则 $f(a) < g(b)$; 若 $a \cdot > b$, 则 $f(a) > g(b)$ 。
- 第一个关系从函数的构造直接获得。因为, 若 $a \cdot = b$, 则既有从 f_a 到 g_b 的弧, 又有从 g_b 到 f_a 的弧。所以, f_a 和 g_b 所能到达的结是全同的。
- 至于 $a \cdot > b$ 和 $a \cdot < b$ 的情形, 只须证明其一。

- 如果 $a \cdot > b$, 则有从 f_a 到 g_b 的弧。也就是, g_b 能到达的任何结 f_a 也能到达。因此, $f(a) \geq g(b)$ 。
 - 我们所需证明的是, 在这种情况下, $f(a) = g(b)$ 不应成立。
 - 我们将指出, 如果 $f(a) = g(b)$, 则根本不存在优先函数。假若 $f(a) = g(b)$, 那么必有如下的回路:



因此有

$$a \cdot > b, \quad a_1 < \cdot = \cdot b, \quad a_1 \cdot > = \cdot b_1, \quad \dots, \\ a_m \cdot > = \cdot b_m, \quad a < \cdot = \cdot b_m$$

对任何优先函数 f' 和 g' 来说，必定有

$$f'(a) > g'(b) \geq f'(a_1) \geq g'(b_1) \geq \dots \geq \\ f'(a_m) \geq g'(b_m) \geq f'(a)$$

从而导致 $f'(a) > f'(a)$ ，产生矛盾。因此，不存在优先函数 f 和 g 。

- 例：取前面文法 $G(E)$

$$(1) \quad E \rightarrow E + T \quad | \quad T$$

$$(2) \quad T \rightarrow T * F \quad | \quad F$$

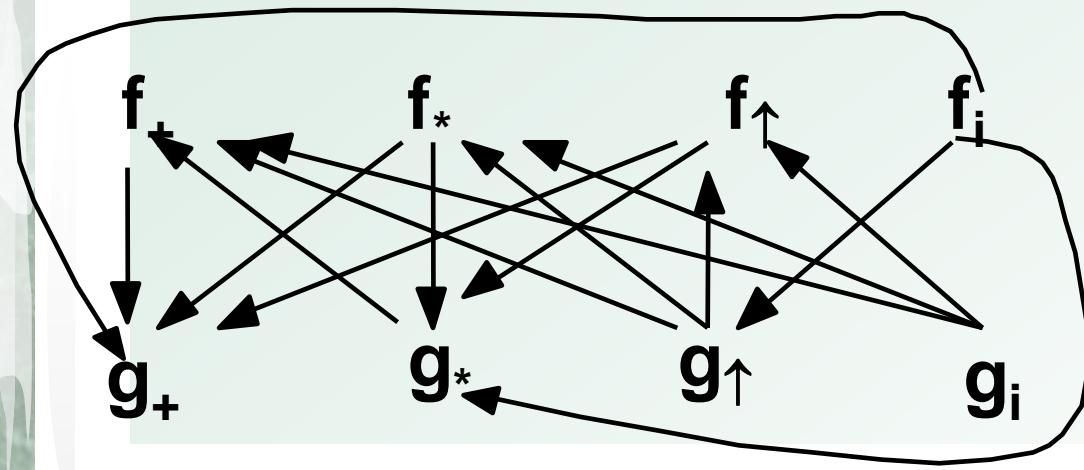
$$(3) \quad F \rightarrow P \uparrow F \quad | \quad P$$

$$(4) \quad P \rightarrow (E) \quad | \quad i$$

的终结符 $+, *, \uparrow, i$



	+	*	↑	i
+				
*				
↑				
i				

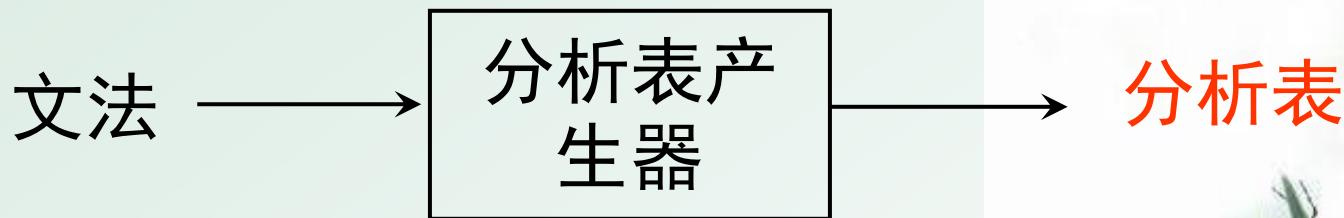


	+	*	↑	i
f	2	4	4	7
g	1	3	6	6

5.3 LR分析法

- LR分析法：1965年 由Knuth提出

➤ 产生分析表



➤ LR分析器工作



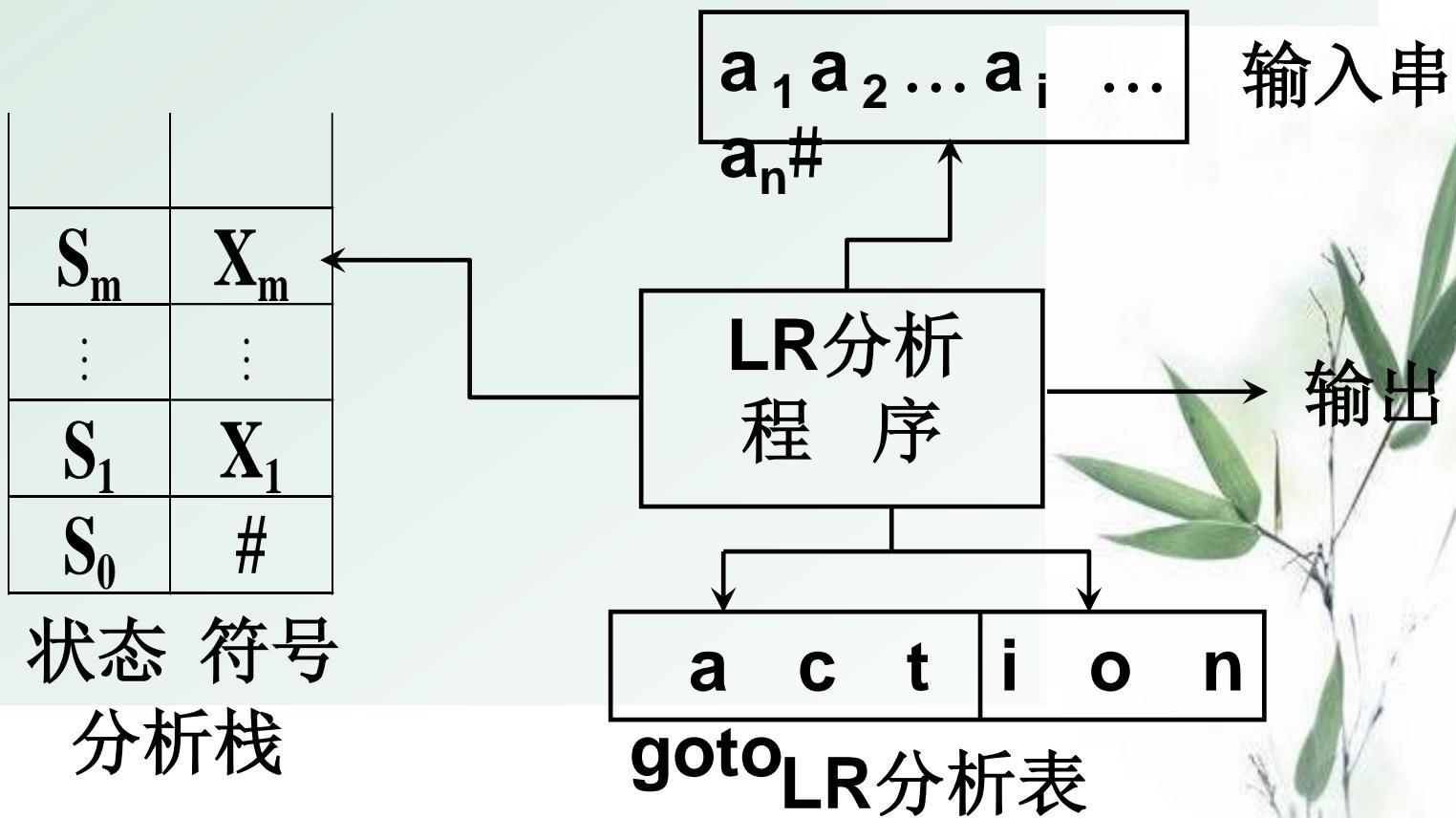
5.3.1 LR分析器

- 规范归约的关键问题是寻找句柄.

- “历史”：已移入符号栈的内容
- “展望”：根据产生式推测未来可能遇到的输入符号
- “现实”：当前的输入符号



- LR分析方法：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



- 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

- 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 $s' = \text{GOTO}[s_{m-r}, A]$ 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

- 分析开始时:

状态 已归约串 输入串

$(s_0, \#, a_1 a_2 \dots a_n \#)$

- 以后每步的结果可以表示为:

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

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

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

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

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

—> $s = \text{GOTO}(s_{m-r}, A)$, r 为 β 的长度, $\beta = X_{m-r+1} \dots X_m$.

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

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

3. 若 $\text{ACTION}(s_m, a_i)$ 为 "接受", 则三元式不再变化, 变化过程终止, 宣布分析成功.
4. 若 $\text{ACTION}(s_m, a_i)$ 为 "报错", 则三元式变化过程终止, 报告错误.

LR分析器示例：

文法G(E) :

- (1) $E \rightarrow E + T$
- (2) $E \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow i$



其LR分析表为：

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

• 假定输入串为*i*i+i*, LR分析器的工作过程:

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

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

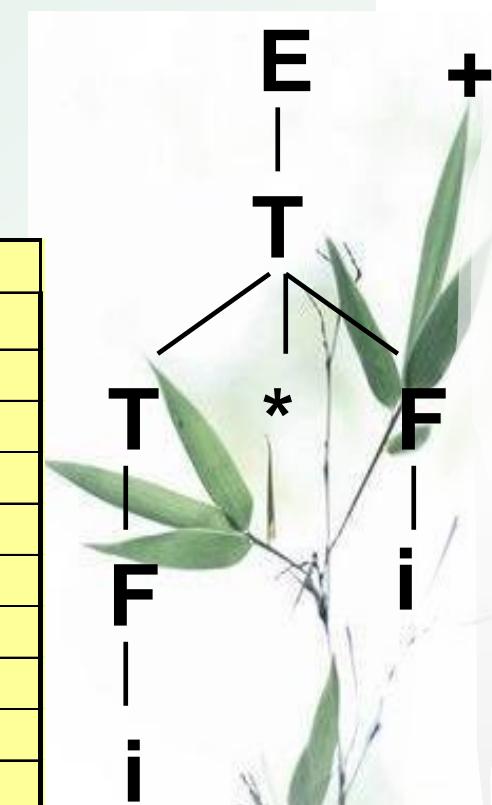
T
—
F
—
i

*

• 假定输入串为*i^{*}i+i*, LR分析器的工作过程:

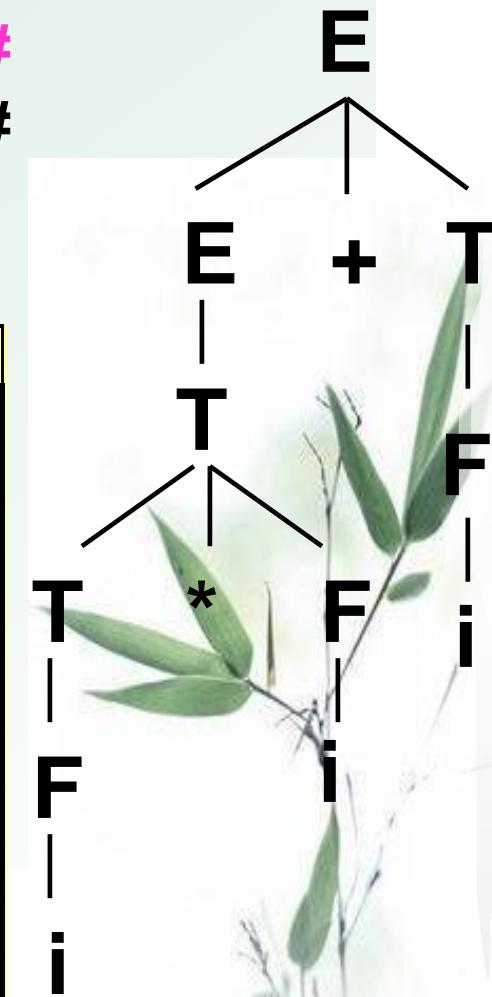
步骤	状态	符号	输入串
(5)027	#T*	i+i#	
(6)0275	#T*i	+i#	
(7)027 <u>10</u>	#T*F	+i#	
(8)02	#T	+i#	
(9)01	#E	+i#	
(10)	016	#E+	i#

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



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

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



- 定义：对于一个文法，如果能够构造一张分析表，使得它的每个入口均是唯一确定的，则这个文法就称为LR文法。
- 定义：一个文法，如果能用一个每步顶多向前检查k个输入符号的LR分析器进行分析，则这个文法就称为LR(k)文法。
- 非LR结构
 - LR文法不是二义的，二义文法肯定不会是LR的。

$S \rightarrow iCtS \mid iCtSeS$	栈	输入
$\cdots iCtS$		$e \cdots \#$



5.3.2 LR(0)项目集族和LR(0)分析表的构造

- 假定 α 是文法G的一个句子，我们称序列

$$\alpha_n, \alpha_{n-1}, \dots, \alpha_0$$

是的一个规范归约，如果此序列满足：

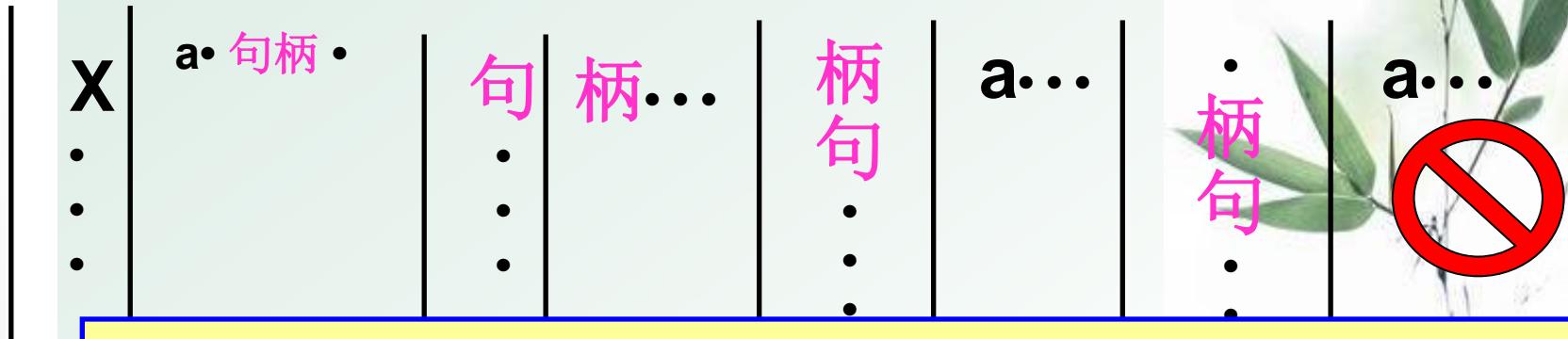
1 $\alpha_n = \alpha$

2 α_0 为文法的开始符号，即 $\alpha_0 = S$

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

5.3.2 LR(0)项目集族和LR(0)分析表的构造

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



栈内永远不会出现句柄之后的符号！

5. 3. 2

LR(0) 项目集族和LR(0) 分析表的构造

- **字的前缀**: 是指字的任意首部, 如字 abc 的前缀有 ε, a, ab, abc
- **活前缀**: 是指规范句型的一个前缀, 这种前缀不含句柄之后的任何符号。即, 对于规范句型 $\alpha\beta\delta$, β 为句柄, 如果 $\alpha\beta=u_1u_2\cdots u_r$, 则符号串 $u_1u_2\cdots u_i$ ($1 \leq i \leq r$) 是 $\alpha\beta\delta$ 的**活前缀**。(δ 必为终结字符串)
- 对于一个文法 G , 可以构造一个DFA, 它能识别 G 的所有活前缀。

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

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

- ☞ $A \rightarrow \alpha .$ 称为"归约项目"
- ☞ 归约项目 $S' \rightarrow \alpha .$ 称为"接受项目"
- ☞ $A \rightarrow \alpha . a\beta$ ($a \in V_T$) 称为"移进项目"
- ☞ $A \rightarrow \alpha . B\beta$ ($B \in V_N$) 称为"待约项目".

•文法G(S')

$$S' \rightarrow E$$

$$E \rightarrow aA | bB$$

$$A \rightarrow cA | d$$

$$B \rightarrow cB | d$$

• 该文法的项目有：

1. $S' \rightarrow \cdot E$

2. $S' \rightarrow E \cdot$

3. $E \rightarrow \cdot aA$

4. $E \rightarrow a \cdot A$

5. $E \rightarrow aA \cdot$

6. $A \rightarrow \cdot cA$

7. $A \rightarrow c \cdot A$

8. $A \rightarrow cA \cdot$

9. $A \rightarrow \cdot d$

10. $A \rightarrow d \cdot$

11. $E \rightarrow \cdot bB$

12. $E \rightarrow b \cdot B$

13. $E \rightarrow bB \cdot$

14. $B \rightarrow \cdot cB$

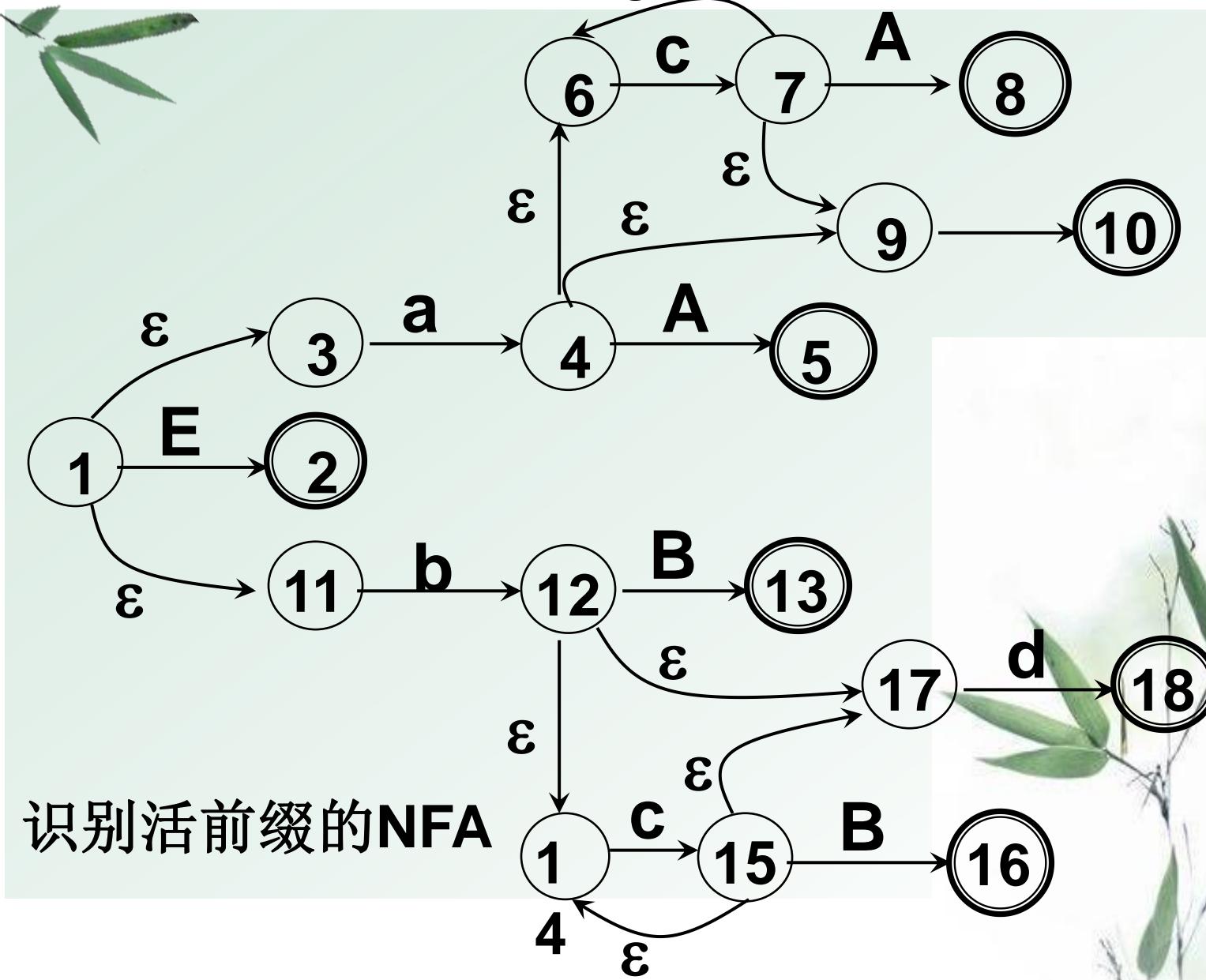
15. $B \rightarrow c \cdot B$

16. $B \rightarrow cB \cdot$

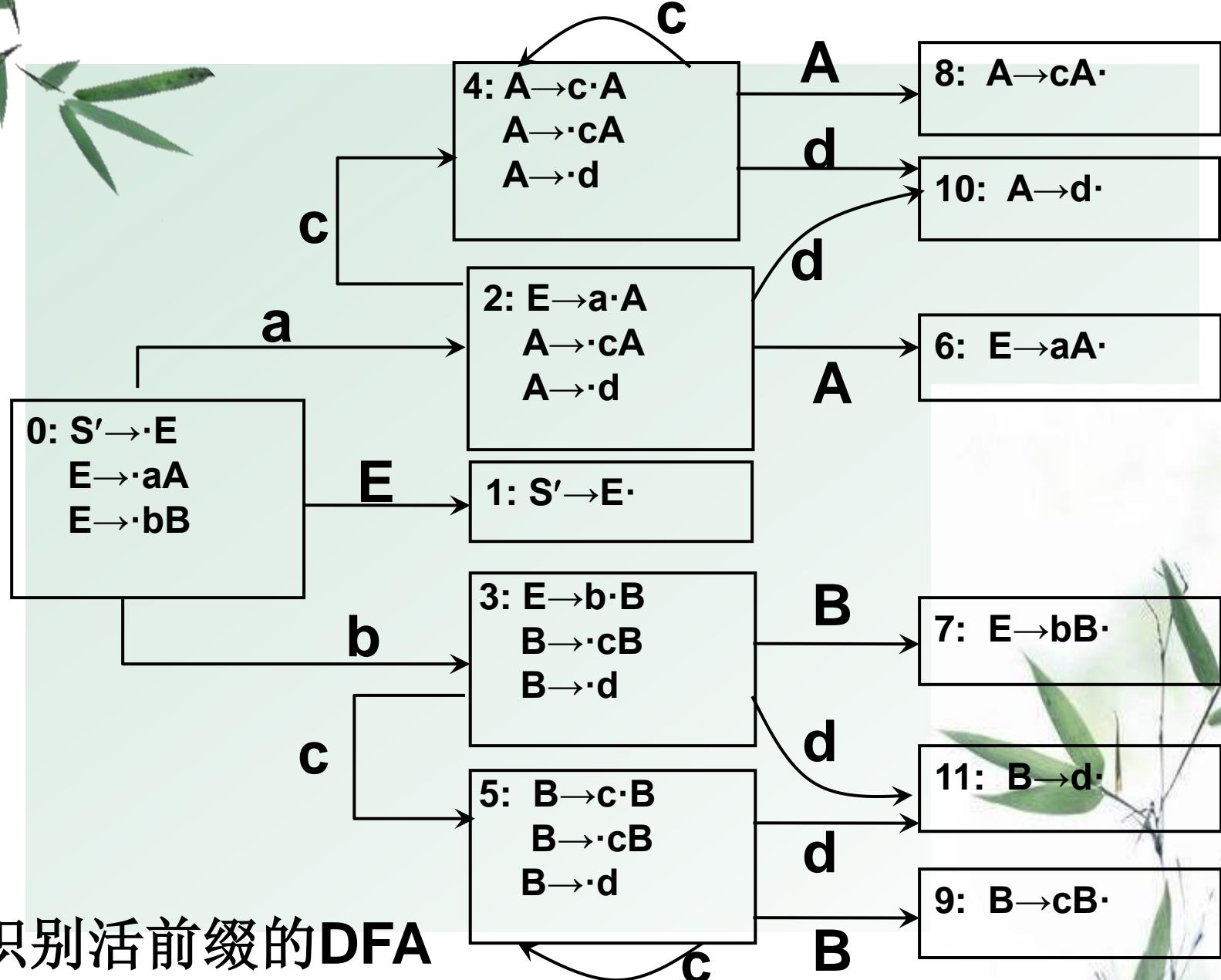
17. $B \rightarrow \cdot d$

18. $B \rightarrow d \cdot$

- 构造识别文法所有活前缀的NFA方法
 1. 若状态 i 为 $X \rightarrow X_1 \dots X_{i-1} \cdot X_i \dots X_n$,
状态 j 为 $X \rightarrow X_1 \dots X_{i-1} X_i \cdot X_{i+1} \dots X_n$,
则从状态 i 画一条标志为 X_i 的有向边到状态 j ;
 2. 若状态 i 为 $X \rightarrow \alpha \cdot A\beta$, A 为非终结符,
则从状态 i 画一条 ε 边到所有状态 $A \rightarrow \cdot \gamma$ 。
- 把识别文法所有活前缀的NFA确定化。



识别活前缀的NFA



识别活前缀的DFA

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

有效项目

- 我们说项目 $A \rightarrow \beta_1 \cdot \beta_2$ 对活前缀 $\alpha\beta_1$ 是有效的，其条件是存在规范推导

$$S' \xrightarrow[R]{*} \alpha A \omega \xrightarrow[R]{} \alpha \beta_1 \beta_2 \omega$$

- 在任何时候，分析栈中的活前缀 $x_1 x_2 \dots x_m$ 的有效项目集正是栈顶状态 S_m 所代表的那个集合。也正是从识别活前缀的DFA的初态出发，读出 $x_1 x_2 \dots x_m$ 后到达的那个项目集（状态）。

- 结论：若项目 $A \rightarrow^* \alpha . B\beta$ 对活前缀 $\eta = \delta\alpha$ 是有效的且 $B \rightarrow^* \gamma$ 是一个产生式，则项目 $B \rightarrow^* . \gamma$ 对 $\eta = \delta\alpha$ 也是有效的。

$$S' \xrightarrow[R]{*} \delta A \omega \xrightarrow[R]{*} \delta \alpha B \beta \omega$$

设 $\beta \omega \xrightarrow[R]{*} \varphi \omega$ ，那么

$$S \xrightarrow[R]{*} \delta A \omega \xrightarrow[R]{*} \delta \alpha B \beta \omega \xrightarrow[R]{*} \delta \alpha B \varphi \omega \xrightarrow[R]{*} \delta \alpha \gamma \varphi \omega$$

所以， $B \rightarrow^* . \gamma$ 对 $\eta = \delta\alpha$ 也是有效的。

- 文法G(S')

$$\begin{aligned}S' &\rightarrow E \\E &\rightarrow aA \mid bB \\A &\rightarrow cA \mid d \\B &\rightarrow cB \mid d\end{aligned}$$

- 考慮：

项目： $B \rightarrow c.B$ $B \rightarrow .cB$ $B \rightarrow .d$
活前缀： bc

$$S' \Rightarrow E \Rightarrow bB \Rightarrow bcB$$

$$S' \Rightarrow E \Rightarrow bB \Rightarrow bcB \Rightarrow bccB$$

$$S' \Rightarrow E \Rightarrow bB \Rightarrow bcB \Rightarrow bcd$$

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

$$S' \xrightarrow[R]{*} \alpha A \omega \xrightarrow[R]{} \alpha \beta_1 \beta_2 \omega$$



LR(0) 项目集规范族的构造

- 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' ，它包含了整个 G ，但它引进了一个不出现在 G 中的非终结符 S' ，并加进一个新产生式 $S' \rightarrow S$ ，而这个 S' 是 G' 的开始符号。那么，我们称 G' 是 G 的**拓广文法**。这样，便会有一个仅含项目 $S' \rightarrow S.$ 的状态，这就是唯一的“接受”态。

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

2. 若状态i为 $X \rightarrow \alpha . A\beta$ ，A为非终结符，
则从状态i画一条 ϵ 边到所有状态 $A \rightarrow . \gamma$ 。

3. 若 $A \rightarrow \alpha . B\beta$ 属于CLOSURE(I)，那么，对于任何关于B的产生式 $B \rightarrow \gamma$ ，项目 $B \rightarrow . \gamma$ 也属于CLOSURE(I)；

4. 重复执行以上步骤直到CLOSURE(I)不再变化。

P50: NFA确定化

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

- i) 若 $s \in I$ ，则 $s \in \epsilon$ -closure(I)；
- ii) 若 $s \in I$ ，则从s出发经过任意条 ϵ 弧而能到达的任何状态 s' 都属于 ϵ -closure(I)

$$\epsilon\text{-closure}(I) = I \cup \{s' \mid \text{从某个 } s \in I \text{ 出发经过任意条 } \epsilon \text{ 弧能到达 } s'\}$$

- 为了识别活前缀，我们定义一个状态转换函数 G_0 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 $G_0(I, X)$ 定义为：

$$G_0(I, X) = \text{CLOSURE}(J)$$

其中

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

- 直观上说，若 I 是对某个活前缀 γ 有效的项目集，那么， $G_0(I, X)$ 便是对 γX 有效的项目集。

■P50：设 a 是 Σ 中的一个字符，定义

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

其中， J 为 I 中的某个状态出发经过一条 a 弧而到达的状态集合。

文法G(S')

$$S' \rightarrow E$$

$$E \rightarrow aA \mid bB$$

$$A \rightarrow cA \mid d$$

$$B \rightarrow cB \mid d$$

- $I_0 = \{S' \rightarrow \cdot E, \quad E \rightarrow \cdot aA, \quad E \rightarrow \cdot bB\}$

$$\begin{aligned} GO(I_0, E) &= \text{closure}(J) = \text{closure}(\{S' \rightarrow E \cdot\}) \\ &= \{S' \rightarrow E \cdot\} = I_1 \end{aligned}$$

$$\begin{aligned} GO(I_0, a) &= \text{closure}(J) = \text{closure}(\{E \rightarrow a \cdot A\}) \\ &= \{E \rightarrow a \cdot A, \quad A \rightarrow \cdot cA, \quad A \rightarrow \cdot d\} = I_2 \end{aligned}$$

$$\begin{aligned} GO(I_0, b) &= \text{closure}(J) = \text{closure}(\{E \rightarrow b \cdot B\}) \\ &= \{E \rightarrow b \cdot B, \quad B \rightarrow \cdot cB, \quad B \rightarrow \cdot d\} = I_3 \end{aligned}$$

- 构造文法G的拓广文法G'的LR(0)项目集规范族算法：

```
PROCEDURE ITEMSETS(G') ;
```

```
BEGIN
```

```
C := {CLOSURE({S' → · S})} ;
```

```
REPEAT
```

```
FOR C中每个项目集I和G'的每个符号X DO
```

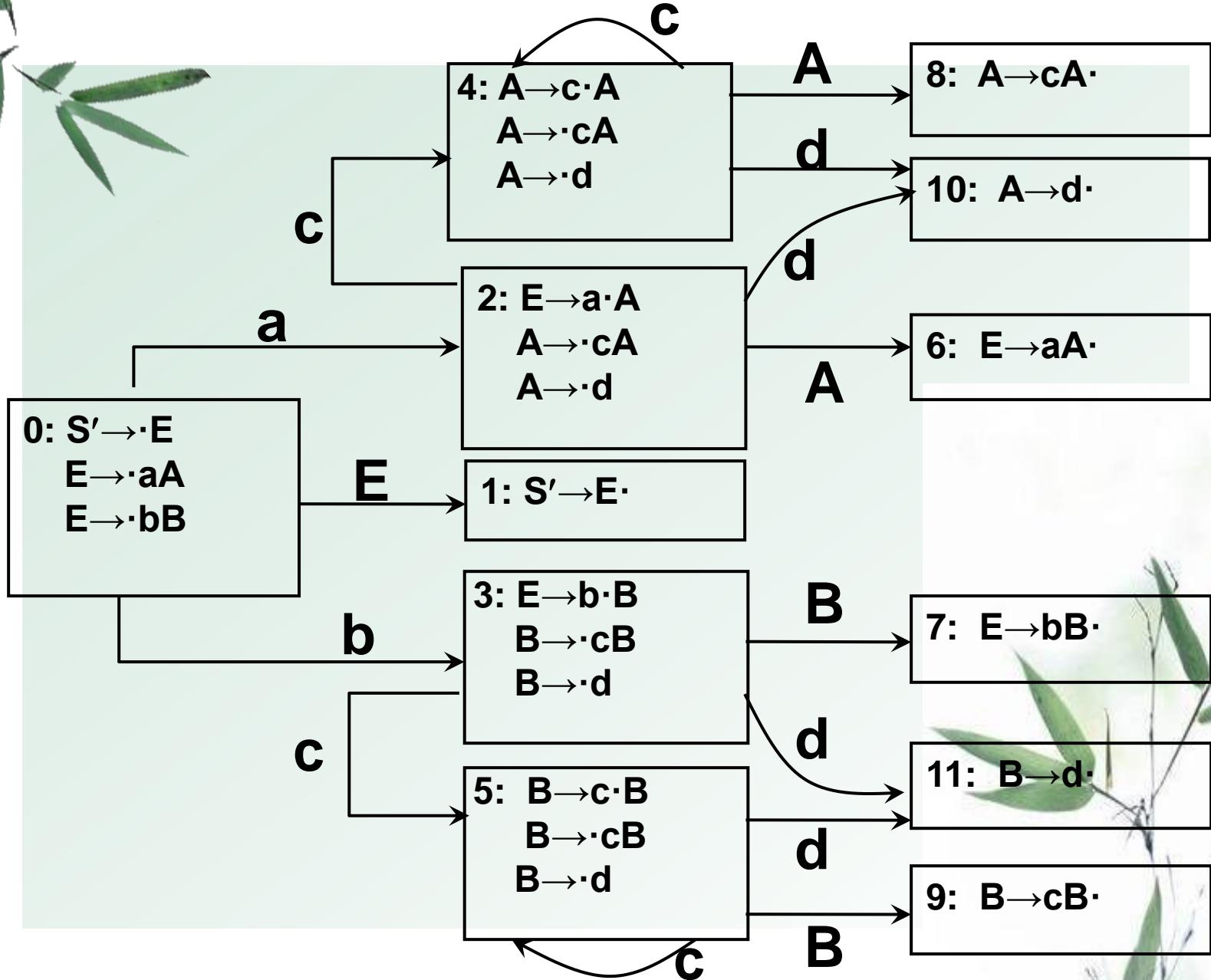
```
IF G0(I, X)非空且不属于C THEN
```

```
把G0(I, X)放入C族中；
```

```
UNTIL C 不再增大
```

```
END
```

- 转换函数G0把项目集连接成一个DFA转换图.



LR(0)分析表的构造

- 假若一个文法G的拓广文法G'的活前缀识别自动机中的每个状态(项目集)不存在下述情况：
 - 既含移进项目又含归约项目，
 - 含有多个归约项目则称G是一个**LR(0)**文法。

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。

- 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”。

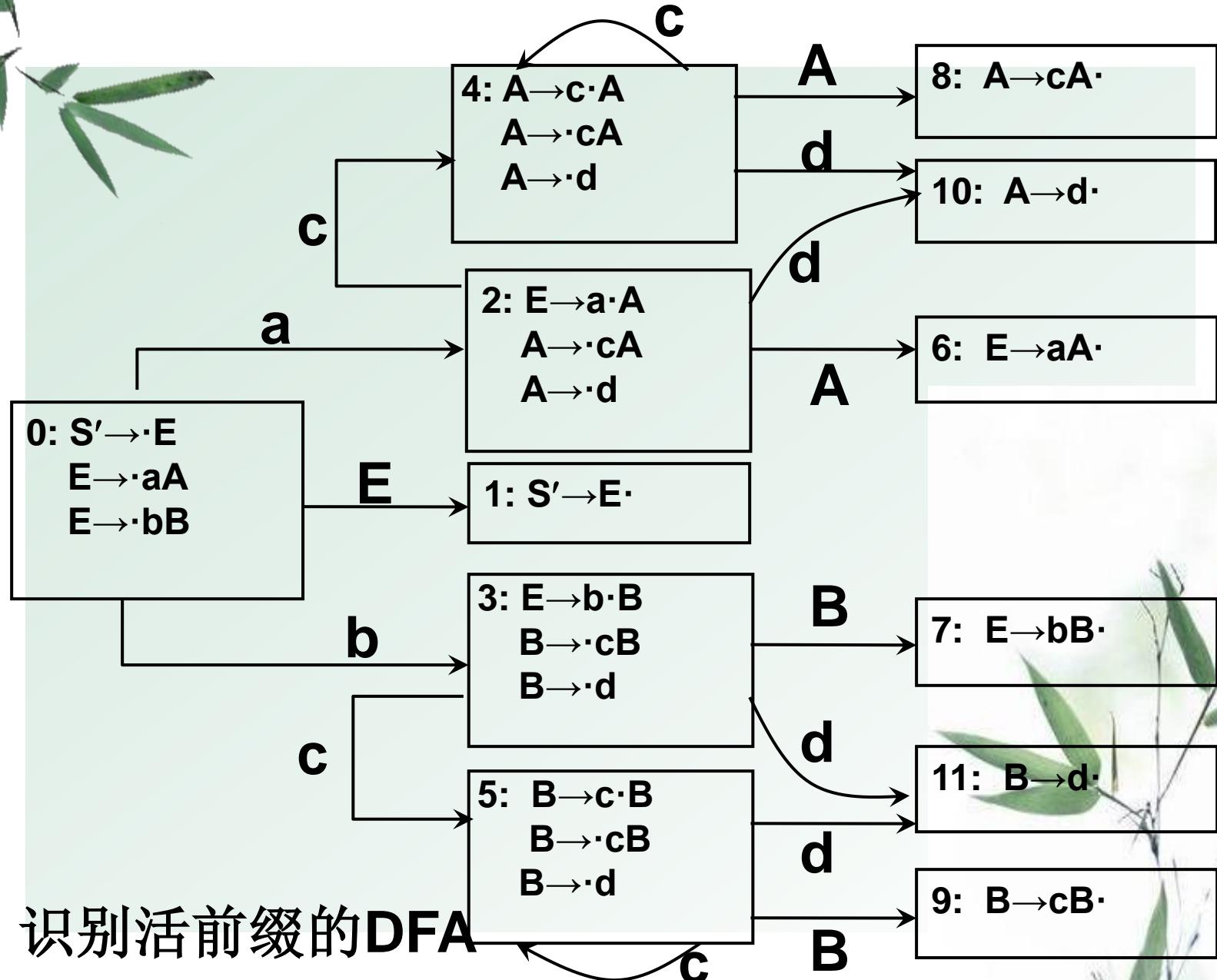
•文法G(S')

$S' \rightarrow E$

$E \rightarrow aA|bB$

$A \rightarrow cA|d$

$B \rightarrow cB|d$



• LR(0)分析表为

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				
6	r1	r1	r1	r1	r1			9
7	r2	r2	r2	r2	r2			
8	r3	r3	r3	r3	r3			
9	r5	r5	r5	r5	r5			
10	r4	r4	r4	r4	r4			
11	r6	r6	r6	r6	r6			

- 例：按上表对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	02444 <u>10</u>	#acccd	#
7	024448	#acccA	#
8	02448	#accA	#
9	0248	#acA	#
10	026	#aA	#
11	01	#E	#

5.3.3 SLR分析表的构造

- LR(0)文法太简单，没有实用价值。
- 假定一个LR(0)规范族中含有如下的一个项目集(状态) $I = \{X \rightarrow \alpha \cdot b\beta, A \rightarrow \alpha \cdot, B \rightarrow \alpha \cdot\}$ 。 $\text{FOLLOW}(A)$ 和 $\text{FOLLOW}(B)$ 的交集为 \emptyset ，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 $a \in \text{FOLLOW}(A)$ ，用产生式 $A \rightarrow \alpha$ 进行归约；
 3. 若 $a \in \text{FOLLOW}(B)$ ，用产生式 $B \rightarrow \alpha$ 进行归约；
 4. 此外，报错。

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

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G0.
- 然后使用C和G0，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 $G0(I_k, a) = I_j$, a 为终结符, 则置 $ACTION[k, a]$ 为“ s_j ”;
 2. 若项目 $A \rightarrow \alpha \cdot$ 属于 I_k , 那么, 对任何终结符 a ,
 $a \in FOLLOW(A)$, 置 $ACTION[k, a]$ 为“ r_j ”; 其中, 假定
 $A \rightarrow \alpha$ 为文法 G' 的第 j 个产生式;
 3. 若项目 $S' \rightarrow S \cdot$ 属于 I_k , 则置 $ACTION[k, #]$ 为“ acc ”;
 4. 若 $G0(I_k, A) = I_j$, A 为非终结符, 则置 $GOTO[k, A] = j$;
 5. 分析表中凡不能用规则1至4填入信息的空白格均置上“出错标志”。

- 按上述方法构造出的ACTION与GOTO表如果不含多重入口，则称该文法为**SLR(1)**文法。
- 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.

- 例5.11 考察下面的拓广文法:

- (0) $S' \rightarrow E$
- (1) $E \rightarrow E + T$
- (2) $E \rightarrow T$
- (3) $T \rightarrow T^* F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow i$



- 这个文法的LR(0)项目集规范族为：

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

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

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

$I_3: T \rightarrow F \cdot$

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

$I_5: F \rightarrow i \cdot$

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

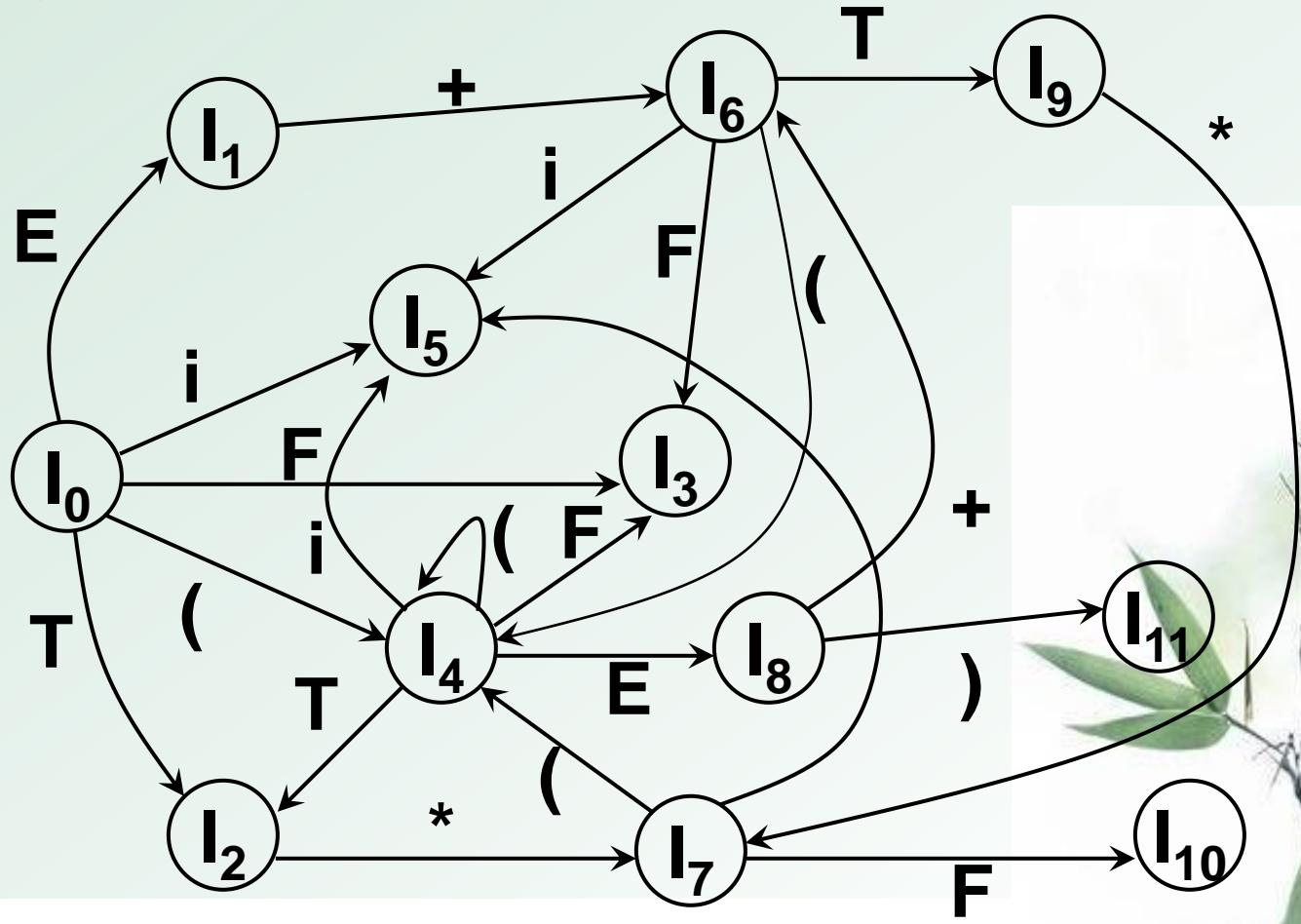
$I_7: T \rightarrow T^* \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot i$

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

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

$I_{10}: T \rightarrow T^* F \cdot$

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



- I_1 、 I_2 和 I_9 都含有“移进—归约”冲突。
- $\text{FOLLOW}(E) = \{\#, \text{ }, +\},$

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

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

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

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			

计算FOLLOW集合所得到的超前符号集合可能大于实际能出现的超前符号集。

- 非SLR文法示例：考慮如下文法：

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



这个文法的LR(0)项目集规范族为：

I₀: S'→·S
S→·L=R
S→·R
S→·*R
L→·i
R→·L

I₁: S'→S·

I₂: S→L·=R
R→L·

I₃: S→R·

I₄: L→*·R
R→·L
L→·*R
L→·i

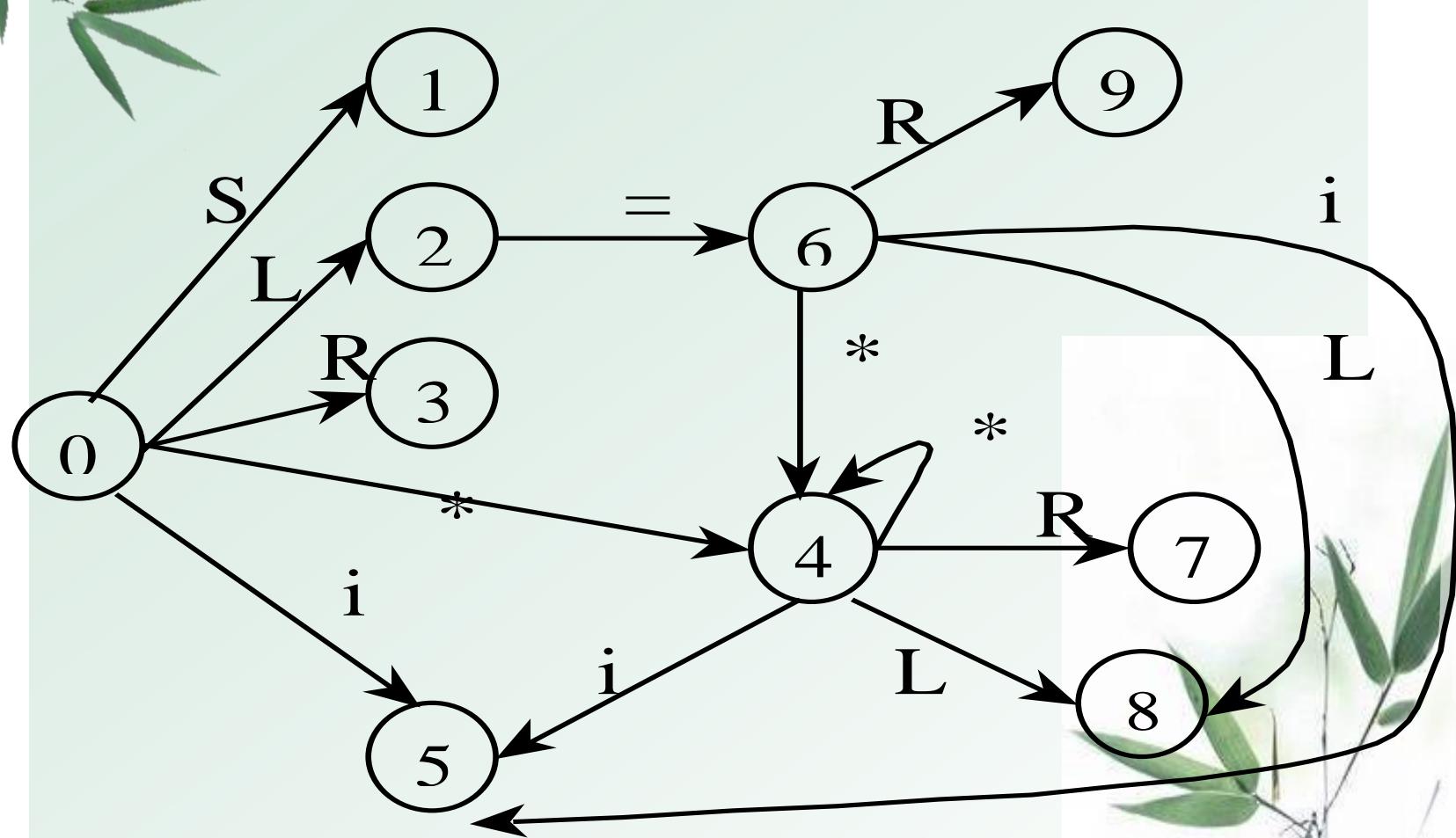
I₅: L→i·

I₆: S→L=·R
R→·L
L→·*R
L→·i

I₇: L→*R·

I₈: R→L·

I₉: S→L=R·



活前缀识别器

- I_2 含有“移进—归约”冲突。
- $\text{FOLLOW}(R) = \{\#, \textcolor{red}{=} \} ,$

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



- 考虑如下文法：

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

不含 “ $R=$ ” 为前缀的规范句型
含 “ $*R=$ ” 为前缀的规范句型

I₂: $S \rightarrow L \cdot = R$
 $R \rightarrow L \cdot$

- 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。

- SLR在方法中, 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“ $\beta A a$ ”的规范句型。因此, 在这种情况下, 用“ $A \rightarrow \alpha$ ”归约不一定合适。

5.3.4 规范LR分析表的构造

- 我们需要重新定义项目，使得每个项目都附带有 k 个终结符。每个项目的一般形式是 $[A \rightarrow \alpha \cdot \beta, a_1 a_2 \dots a_k]$ ，这样的一个项目称为一个LR(k)项目。项目中的 $a_1 a_2 \dots a_k$ 称为它的向前搜索字符串(或展望串)。
- 向前搜索字符串仅对归约项目 $[A \rightarrow \alpha \cdot, a_1 a_2 \dots a_k]$ 有意义。对于任何移进或待约项目 $[A \rightarrow \alpha \cdot \beta, a_1 a_2 \dots a_k]$, $\beta \neq \varepsilon$, 搜索字符串 $a_1 a_2 \dots a_k$ 没有作用。

- 归约项目 $[A \rightarrow \alpha \cdot, a_1 a_2 \dots a_k]$ 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 $a_1 a_2 \dots a_k$ 时，才可以把栈顶上的 α 归约为 A 。
- 我们只对 $k \leq 1$ 的情形感兴趣，向前搜索（展望）一个符号就多半可以确定“移进”或“归约”。
- 形式上我们说一个 LR(1) 项目 $[A \rightarrow \alpha \cdot \beta, a]$ 对于活前缀 γ 是有效的，如果存在规范推导

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

其中，1) $\gamma = \delta \alpha$ ；2) a 是 ω 的第一个符号，或者 a 为 # 而 ω 为 ε 。

- 为构造有效的LR(1)项目集族我们需要两个函数CLOSURE和GO。

- $[A \rightarrow \alpha \cdot B\beta, a]$ 对活前缀 $\gamma = \delta\alpha$ 是有效的，则对于每个形如 $B \rightarrow \xi$ 的产生式，对任何 $b \in \text{FIRST}(\beta a)$ ， $[B \rightarrow \cdot \xi, b]$ 对 γ 也是有效的。
- ◆ 证明：若项目 $[A \rightarrow \alpha \cdot B\beta, a]$ 对 $\gamma = \delta\alpha$ 有效，则有规范推导

$$S \xrightarrow{*} \delta A a \chi \Rightarrow \delta \alpha B \beta a \chi$$

$$\because b \in \text{FIRST}(\beta a)$$

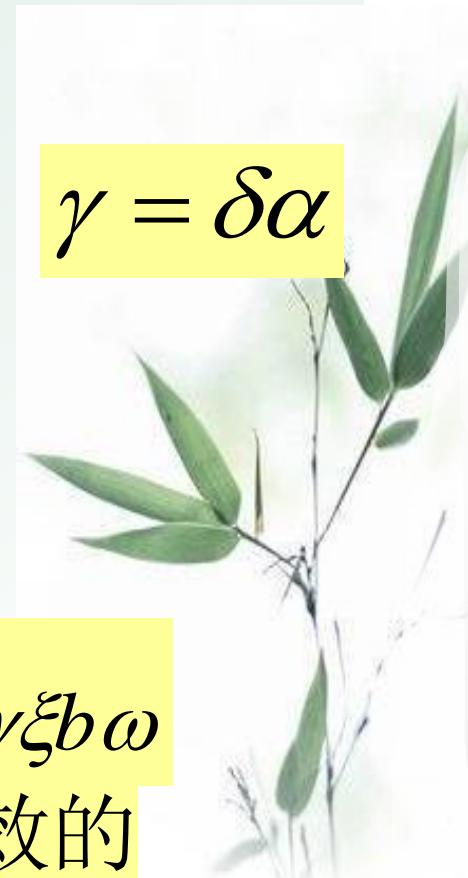
$$\therefore \beta a \chi \xrightarrow{*} b \omega$$

若 $B \rightarrow \xi$ 是产生式

$$\text{则 } S \xrightarrow{*} \delta \alpha B \beta a \chi \xrightarrow{*} \gamma B b \omega \Rightarrow \gamma \xi b \omega$$

\therefore 项目 $[B \rightarrow \cdot \xi, b]$ 对于 γ 是有效的

$$\gamma = \delta\alpha$$



- 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 $[A \rightarrow \alpha \cdot B\beta, a]$ 属于 CLOSURE(I)，
 $B \rightarrow \xi$ 是一个产生式，那么，对于
 $\text{FIRST}(\beta a)$ 中的每个终结符 b ，如果
 $[B \rightarrow \cdot \xi, b]$ 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 $G_0(I, X)$ 定义为：

$$G_0(I, X) = \text{CLOSURE}(J)$$

其中

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

◆ 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

- 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 $[S' \rightarrow \cdot S, \#]$ 的 I_k 的k为分析器的初态。

动作ACTION和状态转换GOTO构造如下：

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“ s_j ”。
2. 若项目 $[A \rightarrow \alpha \cdot, a]$ 属于 I_k , 则置ACTION $[k, a]$ 为“ r_j ”；其中假定 $A \rightarrow \alpha$ 为文法 G' 的第 j 个产生式。
3. 若项目 $[S' \rightarrow S \cdot, \#]$ 属于 I_k , 则置ACTION $[k, \#]$ 为“acc”。
4. 若 $GO(I_k, A) = I_j$, 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，
$$LR(0) \subset SLR \subset LR(1) \subset \text{无二义文法}$$

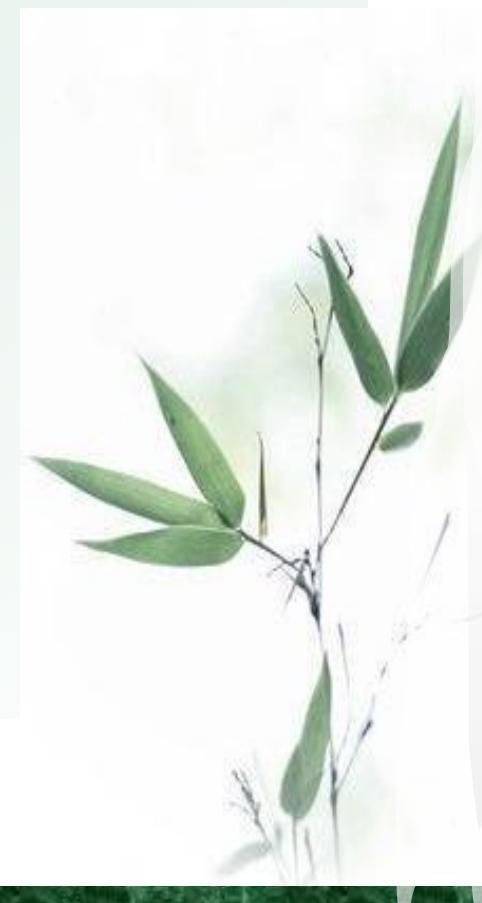
- 例5.13 (5.10)的拓广文法G(S')

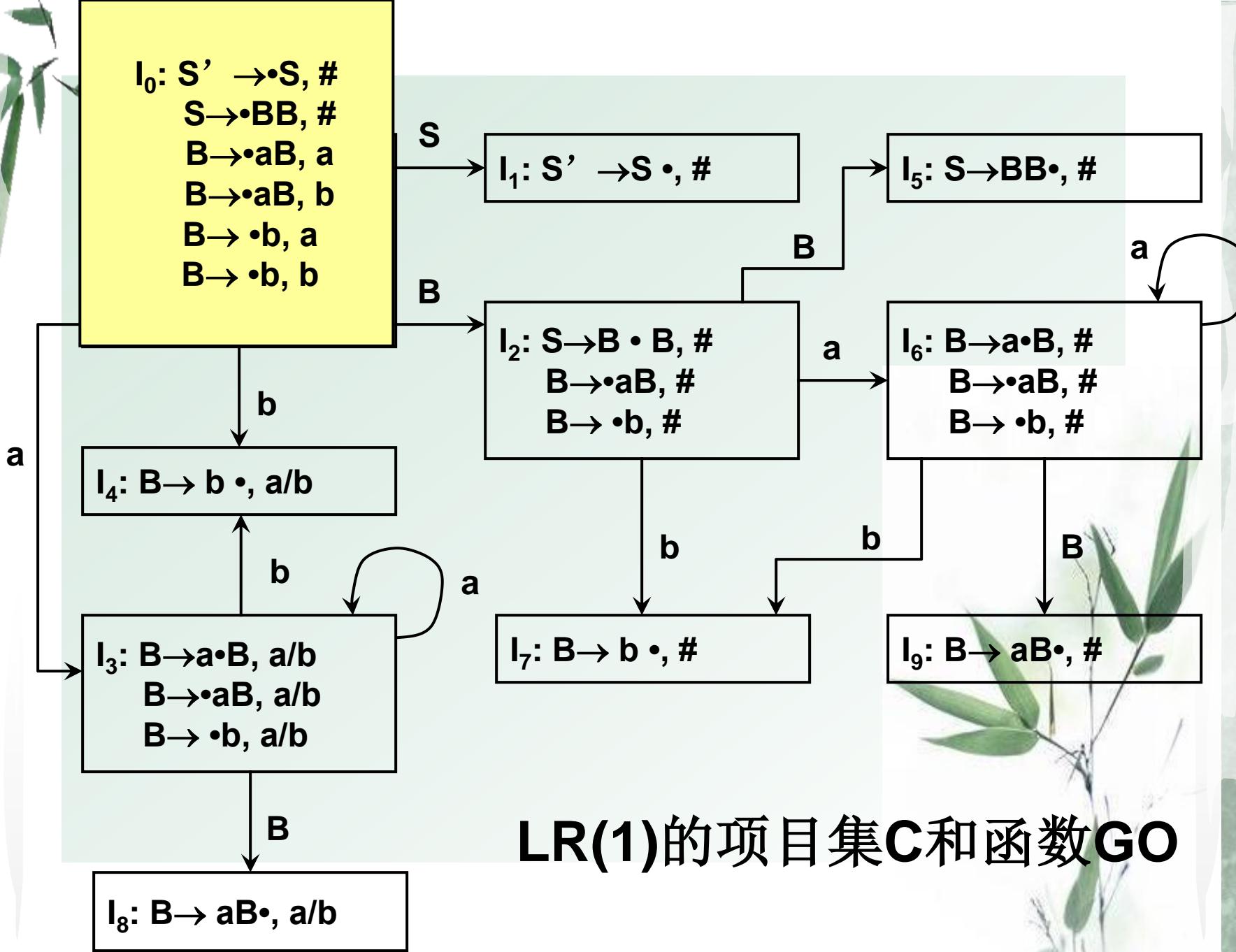
(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$





LR(1)分析表为：

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

- 例：按上表对aabab进行分析

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

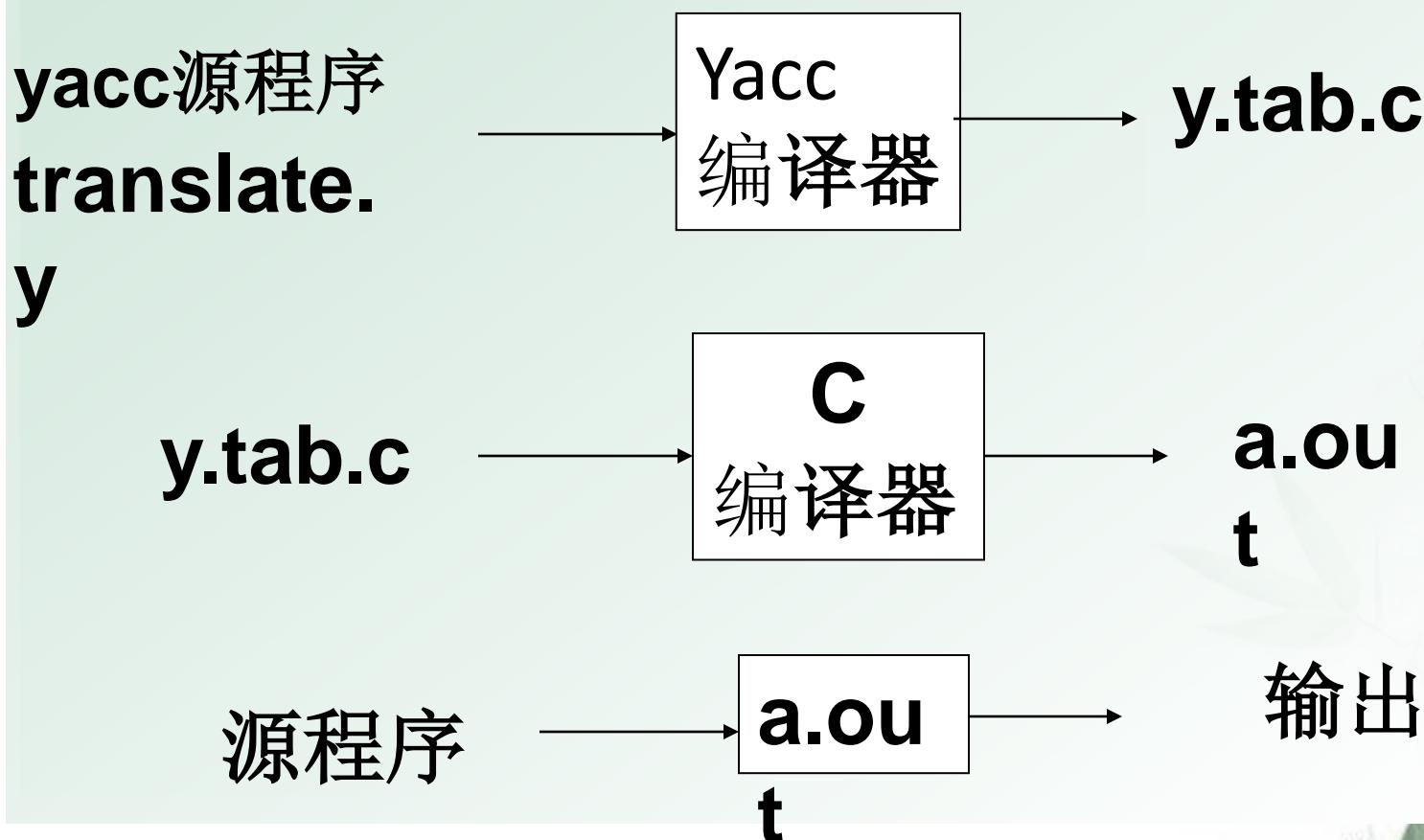
acc

- 例：按上表对abab进行分析

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

5.4 分析器的生成器Yacc

一、用生成器Yacc构造翻译器的过程



二. Yacc源程序有三部分组成

 声明
 %%
 翻译规则
 %%
 C写的支持例程

三. 例4. 21 台式计算器

$$G[E] : \quad E \rightarrow E + T \quad | \quad T$$
$$\qquad\qquad\qquad T \rightarrow T * F \quad | \quad F$$
$$\qquad\qquad\qquad F \rightarrow (E) \quad | \quad \text{digit}$$

读入一个整表达式，计算它的值并输出

声明部分

有任选的两节。

第一节处于分界符%{和%}之间，它是一些普通的C的声明；第二节是文法记号的声明。

翻译规则部分 每条翻译规则由一个文法产生式和有关的语义动作组成。

支持例程部分 一些C写的支持例程。例：词法分析器，错误恢复例程等。

总结：

自顶向下分析

递归预测分析（递归子程序法）

非递归预测分析——LL(1)

注意：首先消除左递归和提取左公因子。

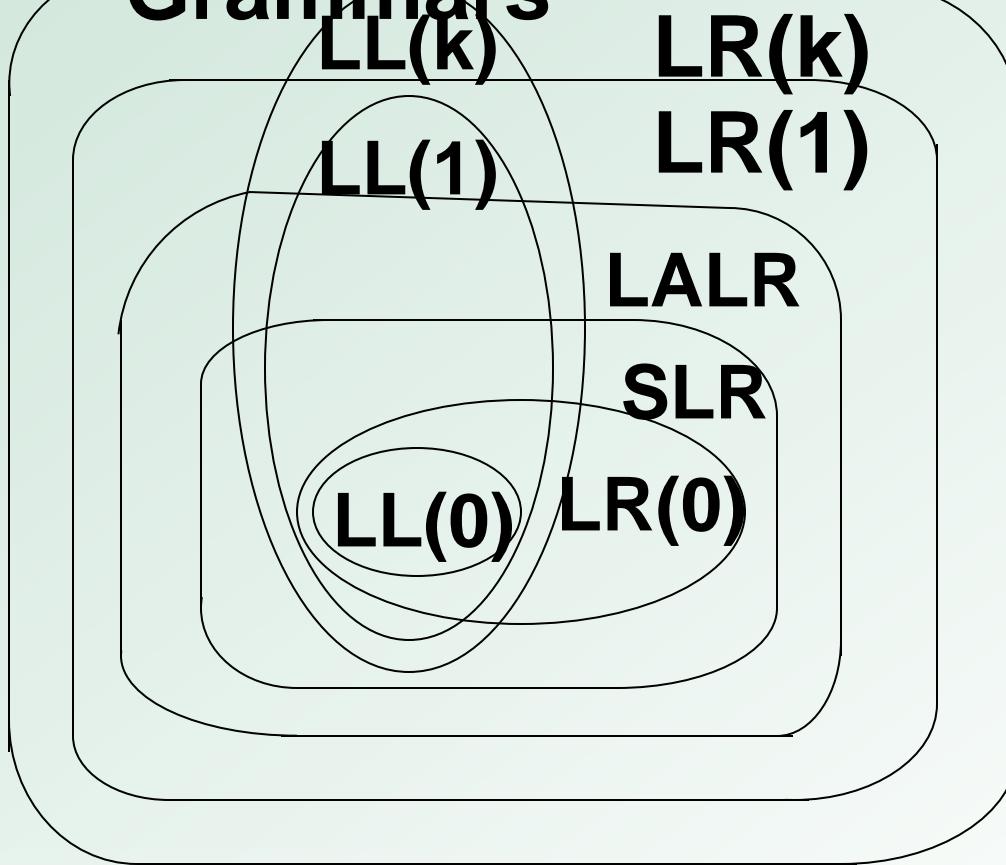
自底向上分析

算符优先分析

**LR分析: LR(0), SLR(1),
LR(1), LALR(1)**

文法类的谱系

Unambiguous Grammars



Ambiguous Grammars