



# 第五章 语法分析—— 自下而上分析

同济大学计算机系

# 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”，直到根结点。

自上而下分析法：

开始符号  $S \xRightarrow{*}$  输入串  $\alpha$  （推导）

自下而上分析法：

输入串  $\alpha \Rightarrow^*$  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**，则将栈顶的这部分替换（**归约**）为该产生式的左部符号。

例. 给定文法 G:

(1)  $S \rightarrow aAcBe$

(2)  $A \rightarrow b$

(3)  $A \rightarrow Ab$

(4)  $B \rightarrow d$

输入串 abbcde 是否为句子?

归约过程如下:



例. 给定文法 G:

(1)  $S \rightarrow aAcBe$

(2)  $A \rightarrow b$

(3)  $A \rightarrow Ab$

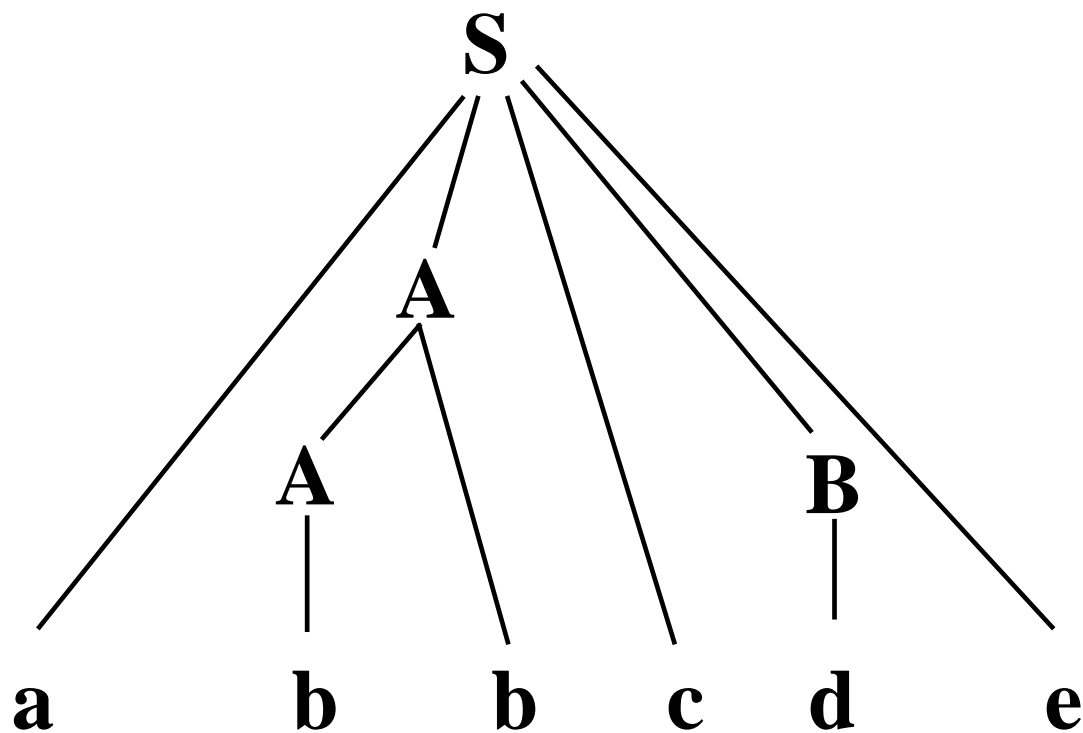
(4)  $B \rightarrow d$       输入串 abbcde 是否为句子?

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	
			b			d	B	B	
	b	A	A		c	c	c	c	
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 $\alpha\beta\delta$ 是文法G的一个句型，如果有

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

则称 $\beta$ 是句型 $\alpha\beta\delta$ 相对于非终结符A的**短语**。

特别地，若  $A \Rightarrow \beta$ ，则称 $\beta$ 是句型 $\alpha\beta\delta$ 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)  $S \rightarrow aAcBe$   
(2)  $A \rightarrow b$   
(3)  $A \rightarrow Ab$   
(4)  $B \rightarrow d$

给出句型aAbcde的短语、直接短语、句柄。

由  $S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde$

$S \Rightarrow aAcBe \Rightarrow aAbcBe \Rightarrow aAbcde$

短语: d, Ab, aAbcde

直接短语: d, Ab

句柄: Ab

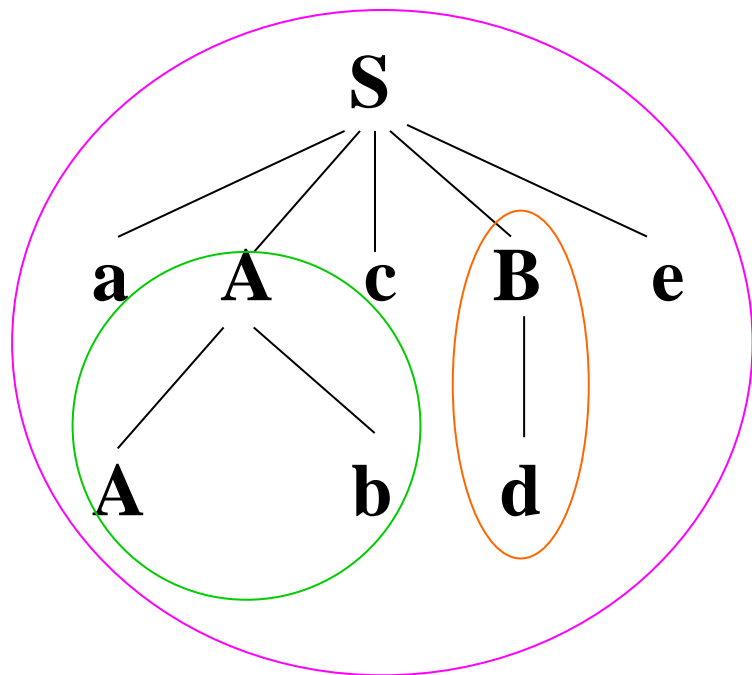
# 句型语法树和句型的短语、直接短语、句柄

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

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

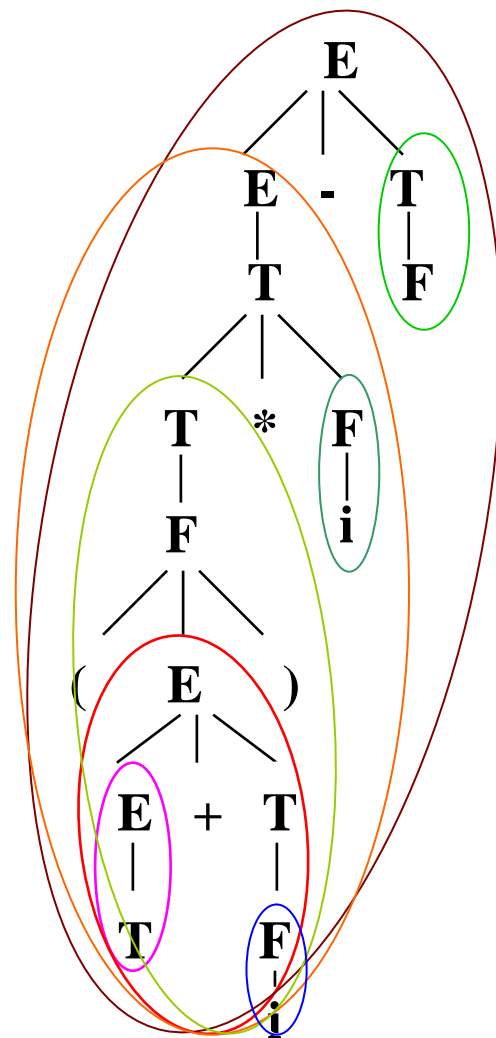
直接短语：d, Ab

句柄：Ab

例. G (E) :  $E \rightarrow E+T \mid E-T \mid T$   
 $T \rightarrow T * F \mid T / F \mid F$   
 $F \rightarrow i \mid (E)$

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:  $E \rightarrow E+E | E^*E | (E) | i$

给出句型 $E+E^*E$ 的句柄

解. (1)  $E \Rightarrow E+\textcolor{red}{E} \Rightarrow E+\textcolor{green}{E^*E}$

$E^*E$ 是句柄

(2)  $E \Rightarrow \textcolor{red}{E^*E} \Rightarrow \textcolor{green}{E+E^*E}$

$E+E$ 是句柄

注: 二义性文法的句柄可能不唯一

# 规范归约

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

(1)  $\alpha_n = \alpha$ ;

(2)  $\alpha_0 = S$ ;

(3) 对任意 $i$ ,  $0 < i \leq n$ ,  $\alpha_{i-1}$  是从 $\alpha_i$  将句柄替换成相应产生式左部符号而得到的

则称该序列是一个规范归约。



最右推导:  $S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde \Rightarrow a \quad b \quad b \quad c \quad d \quad e$

栈

↑

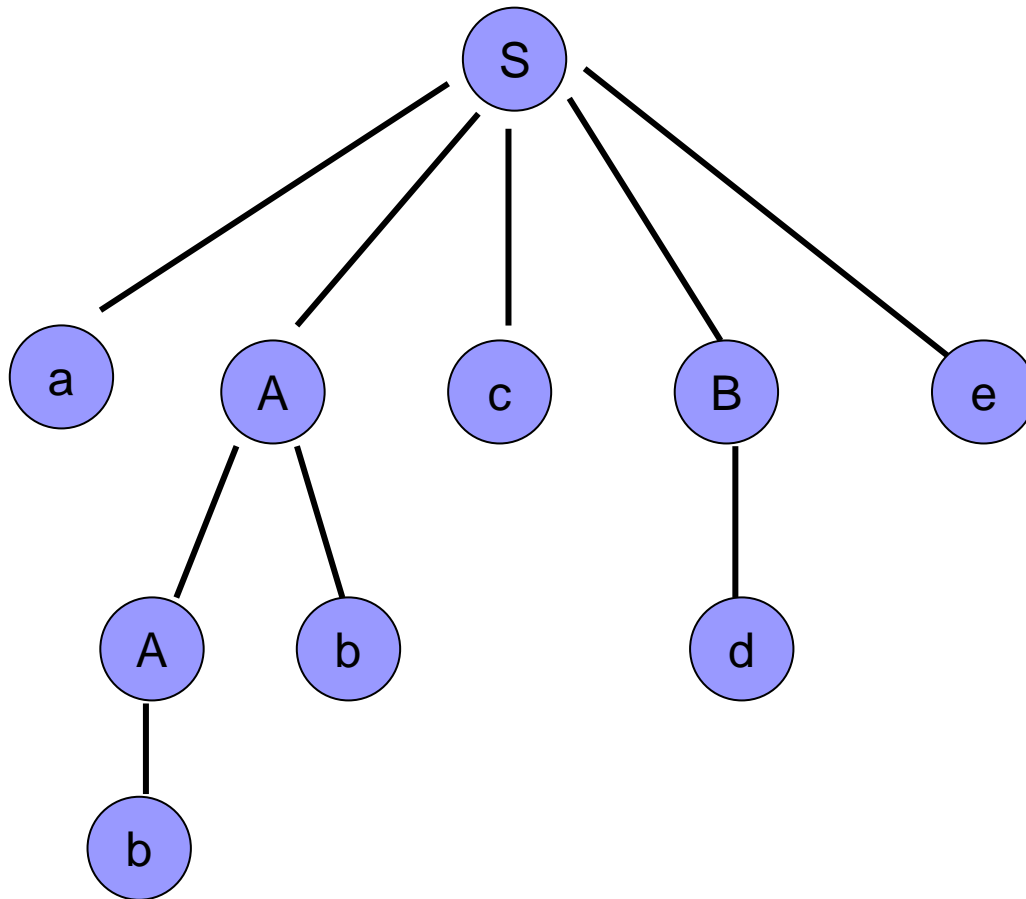
1	2	3	4	5	6	7	8	9	10
								$e$	
						$d \rightarrow B$		$B$	
			$b$		$c$	$c$	$c$	$c$	
	$b \rightarrow A$		$A \rightarrow A$	$A$	$A$	$A$	$A$	$A$	
$a$	$a$	$a$	$a$	$a$	$a$	$a$	$a$	$a$	$S$

$S \rightarrow aAcBe$      $A \rightarrow Ab$

$A \rightarrow b$              $B \rightarrow d$

输入串:  $abbcde$

最左归约:  $a \ b \ b \ c \ d \ e \Rightarrow aAbcde \Rightarrow aAcde \Rightarrow aAcBe \Rightarrow S$



分析树

$S \rightarrow aAcBe$      $A \rightarrow Ab$

$A \rightarrow b$          $B \rightarrow d$

输入串: abbcde

句子abbcde 的规范归约过程如下:

——“剪枝”

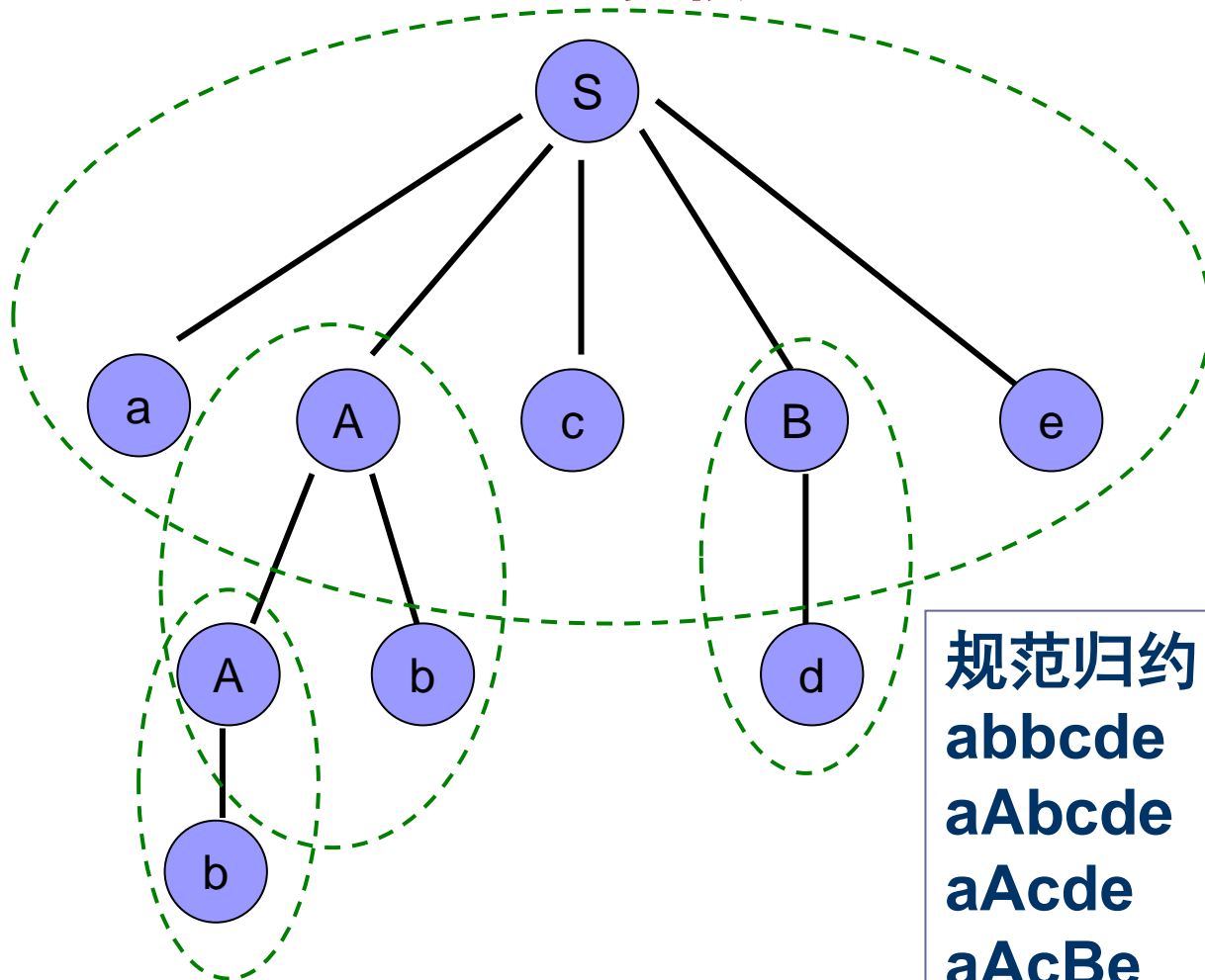
文法G (S)

$S \rightarrow aAcBe$

$A \rightarrow b$

$A \rightarrow Ab$

$B \rightarrow d$



规范归约

abbcde

aAbcde

aAcde

aAcBe

S

归约规则

$A \rightarrow b$

$A \rightarrow Ab$

$B \rightarrow d$

$S \rightarrow aAcBe$

# 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

初始	栈	输入串
	#	w#

最终	栈	输入串
	#S	#

例.  $G: E \rightarrow E+E \mid E * E \mid (E) \mid i$  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 $E \rightarrow i$
3	$\#E*$	$i_2 + i_3 \#$	移进
4	$\#E*i_2$	$+i_3 \#$	移进
5	$\#E * E$	$+i_3 \#$	归约 $E \rightarrow i$
6	$\#E$	$+i_3 \#$	归约 $E \rightarrow E * E$
7	$\#E +$	$i_3 \#$	移进
8	$\#E + i_3$	$\#$	移进
9	$\#E + E$	$\#$	归约 $E \rightarrow i$
10	$\#E$	$\#$	归约 $E \rightarrow E + E$
11	$\#E$	$\#$	接受

# 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

- 发现了一个语法错误,调用出错处理程序

**注：可归约的串在栈顶,不会在内部**

# 语法树的表示——穿线表

## ■ 方法: 在移进-归约过程中自下而上构造句子的语法树

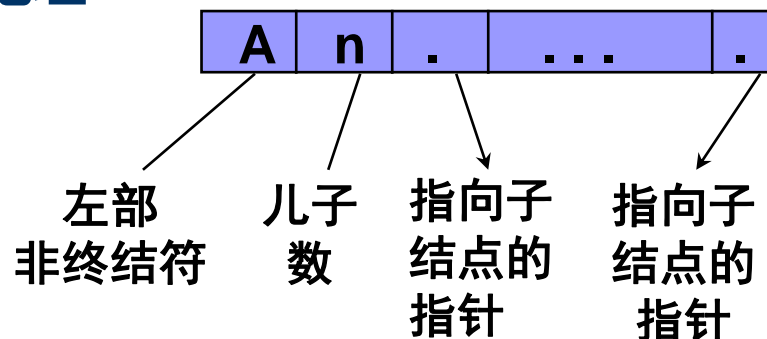
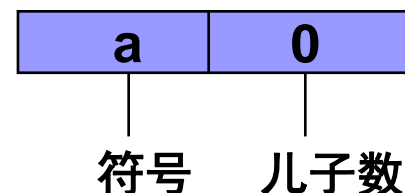
□ 移进符号a时, 构造表示端末结a的数据结构, 其地址与a同时进栈

□ 用  $A \rightarrow X_1 X_2 \cdots X_n$  归约时, 构造新结A的数据结构, 其地址与A同时进栈

□ 接受时, 语法树已经构成, S 及根地址在栈中

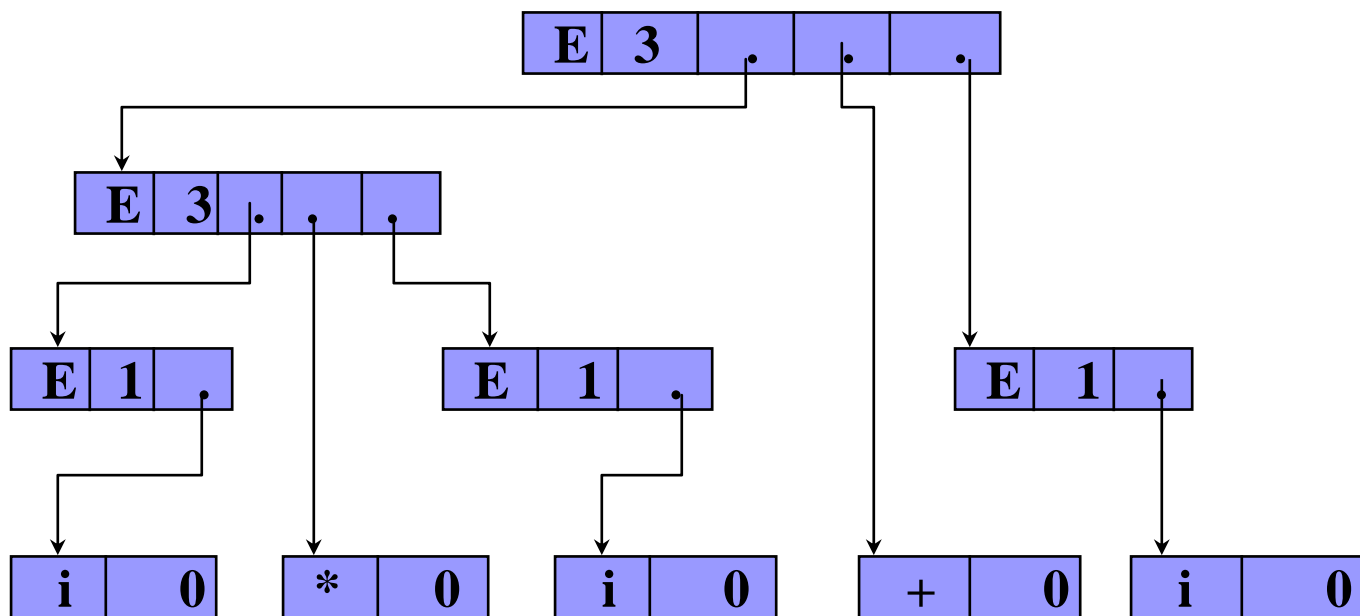
符号	结点地址

符号栈





## 例. $i * i + i$ 的语法树



# 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：

$\square a \lessdot b$       a的优先级低于b

$\square a = b$       a的优先级等于b

$\square a \gtrdot b$       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：

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

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  $\varepsilon$ -产生式,  $a, b \in V_T$ , 算符间的优先关系定义为:

- $a \preceq b$  当且仅当  $G$  含有产生式  $P \rightarrow \dots ab\dots$  或  $P \rightarrow \dots aQb\dots$
- $a \prec b$  当且仅当  $G$  含有产生式  $P \rightarrow \dots aR\dots$  且  $R \xRightarrow{+} b\dots$  或  $R \xRightarrow{+} Qb\dots$
- $a \succ b$  当且仅当  $G$  含有产生式  $P \rightarrow \dots Rb\dots$  且  $R \xRightarrow{+} \dots a$  或  $R \xRightarrow{+} \dots aQ$

# 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

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

则称 G 为**算符优先文法**。

例. 给定文法G:  $E \rightarrow E + E | E * E | (E) | i$

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 $E \rightarrow E + E$ , 且 $E \Rightarrow E * E$ , 所以

$+ \lessdot *$

(2) 因为 $E \rightarrow E * E$ , 且 $E \Rightarrow E + E$ , 所以

$+ \gtrdot *$

**G不是算符优先文法**



例. 文法G: (1)  $E \rightarrow E+T \mid T$       (2)  $T \rightarrow T^*F \mid F$   
 (3)  $F \rightarrow P \uparrow F \mid P$       (4)  $P \rightarrow (E) \mid i$

算符优先关系为:

由(4):  $P \rightarrow (E)$   $\therefore ( = )$

由(1)(2):  $E \rightarrow E+T$ ,  $T \Rightarrow T^*F$   $\therefore + < *$

由(2) (3):  $T \rightarrow T^*F$ ,  $F \Rightarrow P \uparrow F$   $\therefore * < \uparrow$

由(1):  $E \rightarrow E+T$ ,  $E \Rightarrow E+T$   $\therefore + > +$

由(3):  $F \rightarrow P \uparrow F$ ,  $F \Rightarrow P \uparrow F$   $\therefore \uparrow < \uparrow$

由(4):  $P \rightarrow (E)$ ,  $E \Rightarrow E+T$   $\therefore ( < +, + > )$

...

$\therefore$  G为算符优先文法

( # 看作终结符号, 作为句子括号)

	+	*	↑	i	(	)	#
+	➤	⬅	⬅	⬅	⬅	➤	➤
*	➤	➤	⬅	⬅	⬅	➤	➤
↑	➤	➤	⬅	⬅	⬅	➤	➤
i	➤	➤	➤			➤	➤
(	⬅	⬅	⬅	⬅	⬅	H	
)	➤	➤	➤			➤	➤
#	⬅	⬅	⬅	⬅	⬅		H

# 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 $a \preceq b$ 的终结符对。

$a \preceq b$  当且仅当G含有产生式  $P \rightarrow \dots ab\dots$  或  
 $P \rightarrow \dots aQb\dots$

- 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  $P \rightarrow \dots aR \dots$  且  
 $R \xRightarrow{+} b\dots$  或  $R \xRightarrow{+} Qb\dots$

$a > b$  当且仅当G含有产生式  $P \rightarrow \dots Rb\dots$  且  
 $R \xRightarrow{+} \dots a$  或  $R \xRightarrow{+} \dots aQ$

# FIRSTVT(P)和LASTVT(P)

设  $P \in V_N$  , 定义 :

FIRSTVT (P) =

$\{ a \mid P \Rightarrow^+ a \dots \text{ 或 } P \Rightarrow^+ Qa \dots, a \in V_T, Q \in V_N \}$

LASTVT (P) =

$\{ a \mid P \Rightarrow^+ \dots a \text{ 或 } P \Rightarrow^+ \dots aQ, a \in V_T, Q \in V_N \}$

# 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  $\prec$  和  $\succ$  的所有终结符对。

- 假定某个产生式的一个候选形为

...aP...

那么，对任何  $b \in \text{FIRSTVT}(P)$ ，有  $a \prec b$ 。

- 假定某个产生式的一个候选形为

...Pb...

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

# FIRSTVT(P)和LASTVT(P)构造

## ■ FIRSTVT (P) 构造

规则1: 若  $P \rightarrow a \dots$  或  $P \rightarrow Qa \dots$ , 则  $a \in \text{FIRSTVT}(P)$ ;

规则2: 若  $a \in \text{FIRSTVT}(Q)$ , 且  $P \rightarrow Q \dots$ , 则  
 $a \in \text{FIRSTVT}(P)$ 。

## ■ LASTVT (P) 构造

规则1: 若  $P \rightarrow \dots a$  或  $P \rightarrow \dots aQ$ , 则  $a \in \text{LASTVT}(P)$ ;

规则2: 若  $a \in \text{LASTVT}(Q)$ , 且  $P \rightarrow \dots Q$ , 则  
 $a \in \text{LASTVT}(P)$ 。

# FIRSTVT(P)的构造——数据结构

- 二维布尔矩阵**F[P,a]**和符号栈**STACK**

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

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

$$P \rightarrow Q \dots$$

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成为空为止。



例.  $G: S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow T, S \mid S$

求  $\text{FIRSTVT}(S)$ ,  $\text{FIRSTVT}(T)$

$$M = \begin{matrix} & a & \wedge & ( & ) & , \\ \begin{matrix} S \\ T \end{matrix} & \begin{pmatrix} \text{F} & \text{F} & \text{F} & \text{F} & \text{F} \\ \text{F} & \text{F} & \text{F} & \text{F} & \text{F} \end{pmatrix} \end{matrix}$$

T	,
T	(
T	^
T	a

$\text{FIRSTVT}(S) = \{ a, \wedge, ( \}$

$\text{FIRSTVT}(T) = \{ a, \wedge, (, , \}$

类似地可得:  $\text{LASTVT}(S) = \{ a, \wedge, ) \}$

$\text{LASTVT}(T) = \{ a, \wedge, ), , \}$

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

INSERT (P, a);

WHILE STACK非空 DO

BEGIN

把STACK的顶项 (Q, a) 弹出;

FOR 每条形如 $P \rightarrow Q \dots$  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  $P \rightarrow X_1 X_2 \dots X_n$  DO
  FOR  $i := 1$  TO  $n-1$  DO
    BEGIN
      IF  $X_i$  和  $X_{i+1}$  均为终结符 THEN 置  $X_i = X_{i+1}$ 
      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  $X_i = X_{i+2}$ ;
      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  $X_i < a$ ;
      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  $a > X_{i+1}$ 
    END
```

例. G:  $S \rightarrow a \mid \wedge \mid \underline{(T)}$

$\text{FIRSTVT}(S) = \{ a, \wedge, ( \}$

$\text{LASTVT}(S) = \{ a, \wedge, ) \}$

$T \rightarrow \underline{T,S} \mid S$

$\text{FIRSTVT}(T) = \{ a, \wedge, (, , \}$

$\text{LASTVT}(T) = \{ a, \wedge, ), , \}$

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

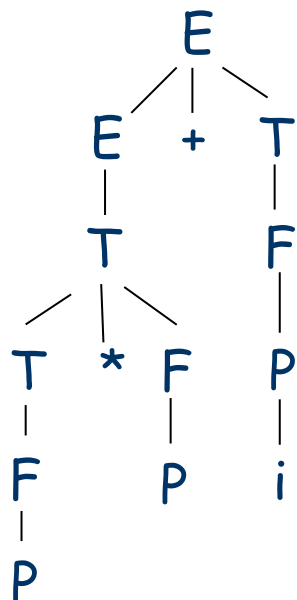
(2)  $T \rightarrow T*F \mid F$

(3)  $F \rightarrow P \uparrow F \mid P$

(4)  $P \rightarrow (E) \mid i$

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:  $P, P*P, P, i, P*P+i$

素短语:  $P*P, i$

最左素短语:  $P*P$

例. 对文法G:

(1)  $E \rightarrow E+T \mid T$

(2)  $T \rightarrow T*F \mid F$

(3)  $F \rightarrow P \uparrow F \mid P$

(4)  $P \rightarrow (E) \mid i$

句型:  $T+F*P+i$

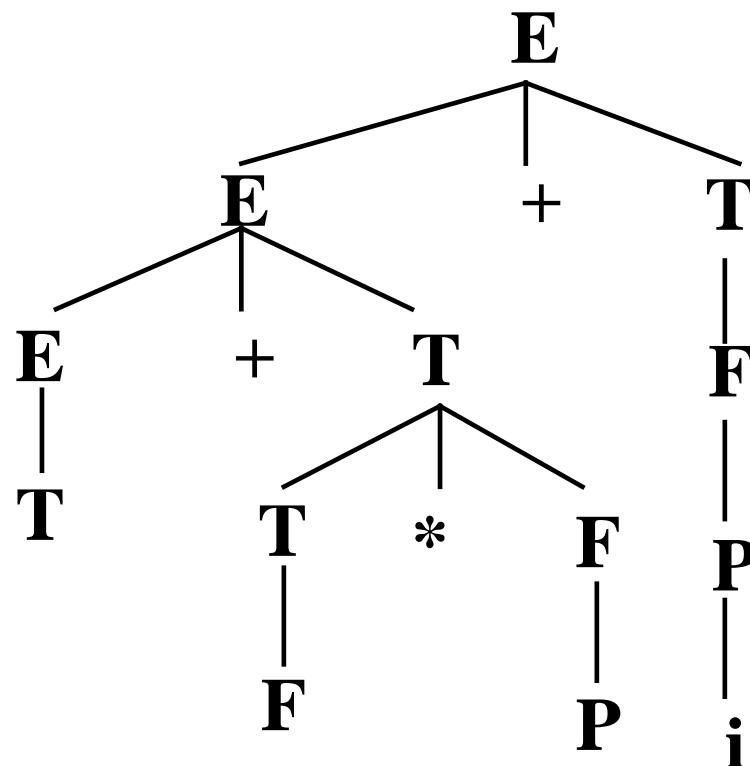
短语:  $T, F, P, i, F*P,$   
 $T+F*P, T+F*P+i$

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:

$$\# N_1 a_1 N_2 a_2 \dots N_n a_n N_{n+1} \#$$

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  $N_j a_j \dots N_i a_i N_{i+1}$

$$a_{j-1} < a_j$$

$$a_j \neq a_{j+1} \neq \dots \neq a_{i-1} \neq a_i$$

$$a_i > a_{i+1}$$

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



# 算符优先分析算法

- 1) 将输入串依次逐个存入符号栈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  $<$  为止；
- 3) 已找到最左素短语 $S_{j+1} \dots S_k$ ，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       $k = 1$ ;  $S[k] := \text{" \# "}$ ;
2      REPEAT
3          把下一个输入字符读进  $a$ ;
4          IF  $S[k] \in V_T$  THEN  $j := k$ ;
5          WHILE  $S[j] > a$  DO
6              BEGIN
7                  REPEAT
8                       $Q := S[j]$ ;
9                      IF  $S[j-1] \in V_T$  THEN  $j := j-1$  ELSE  $j := j-2$ ;
10                     UNTIL  $S[j] < Q$ ;
11                     把  $S[j+1] \dots S[k]$  归约为某个  $N$ ;
12                      $k := j+1$ ;  $S[k] := N$ ;
13             END OF WHILE;
14             IF  $S[j] < a$  OR  $S[j] \neq a$  THEN
15                 BEGIN  $k := k+1$ ;  $S[k] := a$  END
16             ELSE ERROR
17         UNTIL  $a = \text{" \# "}$ 

```

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。

$N$	$\rightarrow$	$X_1$	$X_2$	$\dots$	$X_{k-j}$
		$\updownarrow$	$\updownarrow$		$\updownarrow$
		$S[j+1]$	$S[j+2]$	$\dots$	$S[k]$

例. 对例1中文法G, 符号串 $i*(i+i)$ 的分析过程如下:

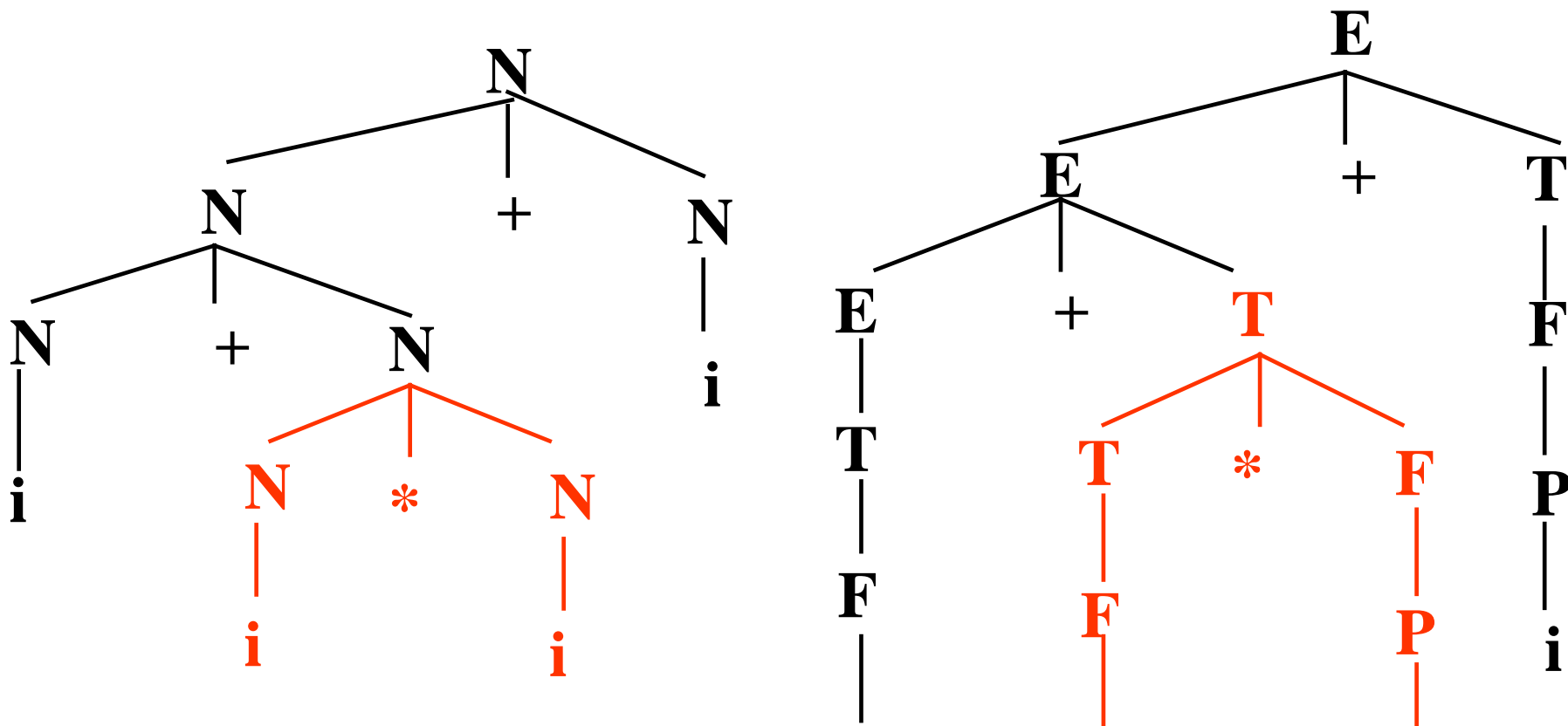
符号栈	关系	输入串	最左素短语
#	<.	$i* (i+i) \#$	
# <b>i</b>	>.	$* (i+i) \#$	i
#N	<.	$* (i+i) \#$	
#N*	<.	$(i+i) \#$	
#N*(	<.	$i+i ) \#$	
#N*( <b>i</b>	>.	$+i) \#$	i
#N*(N	<.	$+ i) \#$	
#N*(N+	<.	$i) \#$	
#N*(N+ <b>i</b>	>.	) #	i
#N*( <b>N+N</b>	>.	) #	N+N
#N*(N	=.	) #	
#N*( <b>N</b> )	>.	#	(N)
# <b>N*N</b>	>.	#	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)  $E \rightarrow E+T \mid T$  (2)  $T \rightarrow T * F \mid F$   
 (3)  $F \rightarrow P \uparrow F \mid P$  (4)  $P \rightarrow (E) \mid i$

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。

# 优先函数

- 在实际实现算符优先分析算法时，考虑到存储开销，一般不采用优先表，而是采用优先函数。
  - 对于文法G的每个终结符 $\theta$ ，函数f和g满足
    - 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  $f(\theta_1) > g(\theta_2)$
- 称f和g是**优先函数**。
- f( $\theta$ ) : 入栈优先函数
- g( $\theta$ ): 比较优先函数
- 实质上，优先函数是优先表的线性化

# 优先函数的特点

- 便于比较，节省空间；
- 原来不存在优先关系的两个终结符，由于与自然数相对应，变成可以比较的。要进行一些特殊的判断。
- 优先函数不唯一，只要存在一对，必存在无穷对优先函数。

- 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

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

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

- 导致如下矛盾：

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

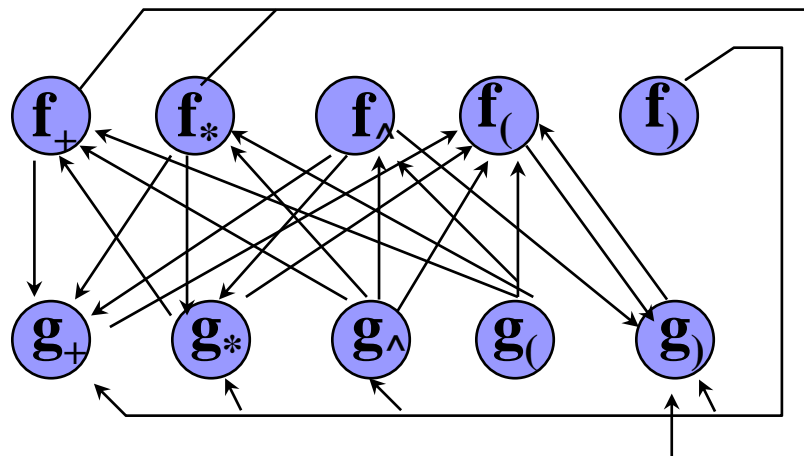


# 优先函数构造方法

- 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：
  - 任  $a \in V_T$  对应于两个结点  $f_a$  和  $g_a$
  - 如果  $a$  的优先级高于或等于  $b$ ，则从  $f_a$  至  $g_b$  画一条有向边  
如果  $a$  的优先级低于或等于  $b$ ，则从  $g_b$  至  $f_a$  画一条有向边
  - 每个结点赋予一个数，该数等于从该结点出发可达结点（包括出发结点本身在内）的个数，分别作为  $f(a)$  和  $f(b)$  等
  - 检查是否矛盾

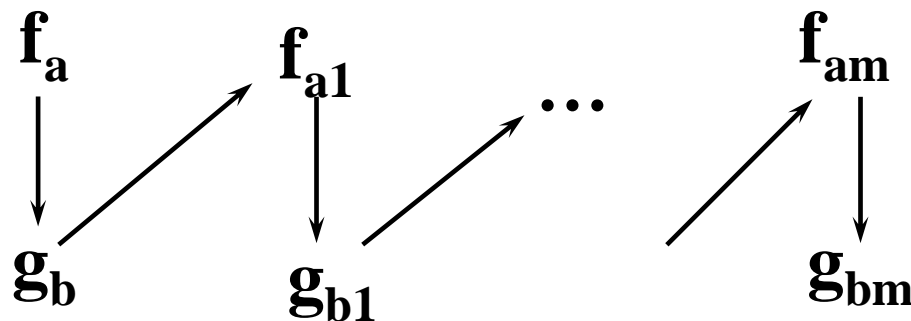
### 例. 表5.1的优先函数为:

	+	*	↑	i	(	)	#
+	➤	⬅	⬅	⬅	⬅	➤	➤
*	➤	➤	⬅	⬅	⬅	➤	➤
↑	➤	➤	⬅	⬅	⬅	➤	➤
i	➤	➤	➤			➤	➤
(	⬅	⬅	⬅	⬅	⬅	⊢	
)	➤	➤	➤			➤	➤
#	⬅	⬅	⬅	⬅	⬅		⊢

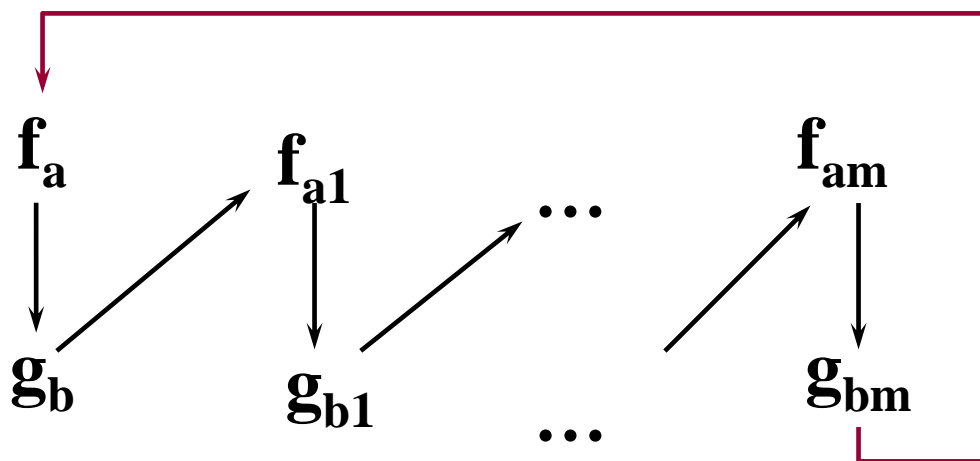


符号 函数	+	*	^	(	)
f	4	6	6	2	9
g	3	5	8	8	2

- 现在必须证明：若 $a = b$ ，则 $f(a)=g(b)$ ；若 $a < b$ ，则 $f(a) < g(b)$ ；若 $a > b$ ，则 $f(a) > g(b)$ 。
- 第一个关系可从函数的构造直接获得。因为，若 $a=b$ ，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 $g_b$ 所能到达的结是相同的。
- 至于 $a < b$ 和 $a > b$ 的情形，只须证明其一。
- 如果 $a > b$ ，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。
- 假若 $f(a) = g(b)$ ，那么必有如下的回路：



因此有

$$a > b, a_1 \leq b, a_1 \geq b_1, \dots, a_m \geq b_m, a \leq b_m$$

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。

# 内容线索

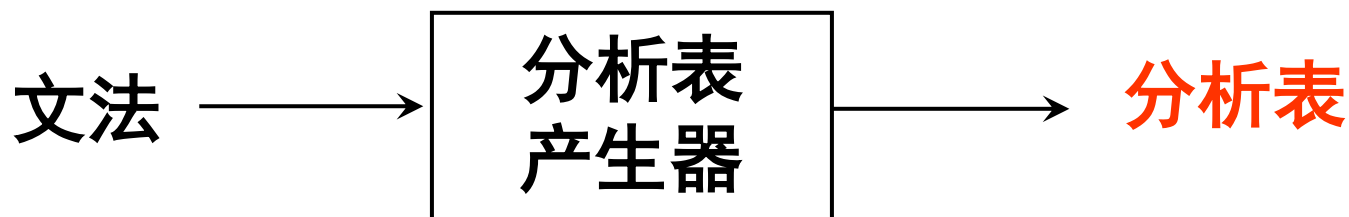
- ✓ 自下而上分析基本问题
- ✓ 规范规约
- ✓ 算符优先分析方法
- LR分析方法

# 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Knuth提出  
产生分析表



## LR分析器工作



# 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



# 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

$S_m$	$X_m$
$\vdots$	$\vdots$
$S_1$	$X_1$
$S_0$	$\#$

状态      符号  
分析栈

$a_1 a_2 \dots a_i \dots a_n \#$

输入串

LR分析  
程 序

输出

action

goto

LR分析表

# 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
  - GOTO[s, X]定义了一个以文法符号为字母表的DFA

# Action[s, a]

## ■ 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 $\beta$ 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：(  $S_0$ , #,  $a_1 a_2 \dots a_n \#$  )

某一步：(  $S_0 S_1 \dots S_m$ ,  $\#X_1 X_2 \dots X_m$ ,  $a_i a_{i+1} \dots a_n \#$  )

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

(  $S_0 S_1 \dots S_m$  **S**,  $\#X_1 X_2 \dots X_m$   **$a_i$** ,  $a_{i+1} \dots a_n \#$  )

若 ACTION [ $S_m$ ,  $a_i$ ] 为 “归约” {  $A \rightarrow \beta$  },

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

则三元式为：

(  $S_0 S_1 \dots S_{m-r}$  **S**,  $\#X_1 X_2 \dots X_{m-r}$  **A**,  $a_i a_{i+1} \dots a_n \#$  )

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

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)  $F \rightarrow i$

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			

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		

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

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

F

i

# LR 文法

- 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。



# 活前缀

- 前缀: 一个字的任意首部。例: 字abc的前缀有 $\epsilon$ , 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 $\alpha\beta\delta$ ,  $\beta$ 为句柄, 如果 $\alpha\beta = u_1u_2\dots u_r$ , 则符号串 $u_1u_2\dots u_i (1 \leq i \leq r)$ 是 $\alpha\beta\delta$ 的活前缀。(  $\delta$ 必为终结字符串)
- 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。

# 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):  $E \rightarrow aA$   
 $A \rightarrow bA|a$

则G的LR (0) 项目有:

$E \rightarrow \cdot aA$        $E \rightarrow a \cdot A$        $E \rightarrow aA \cdot$

$A \rightarrow \cdot bA$        $A \rightarrow b \cdot A$        $A \rightarrow bA \cdot$

$A \rightarrow \cdot a$        $A \rightarrow a \cdot$

## 例. 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. $B \rightarrow d \cdot$   |

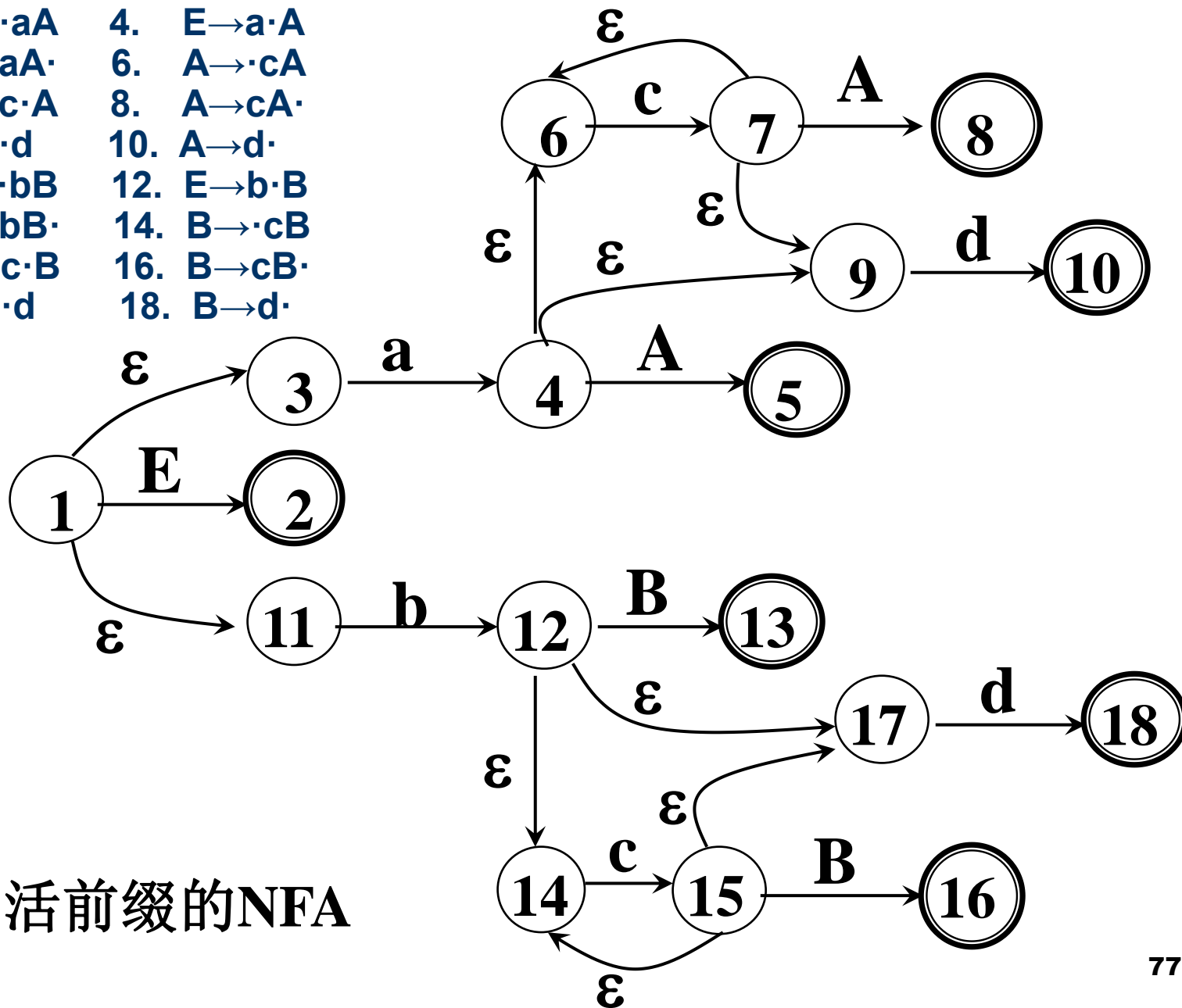
# 方法一：识别活前缀的NFA方法

## ■ 构造识别文法所有活前缀的NFA

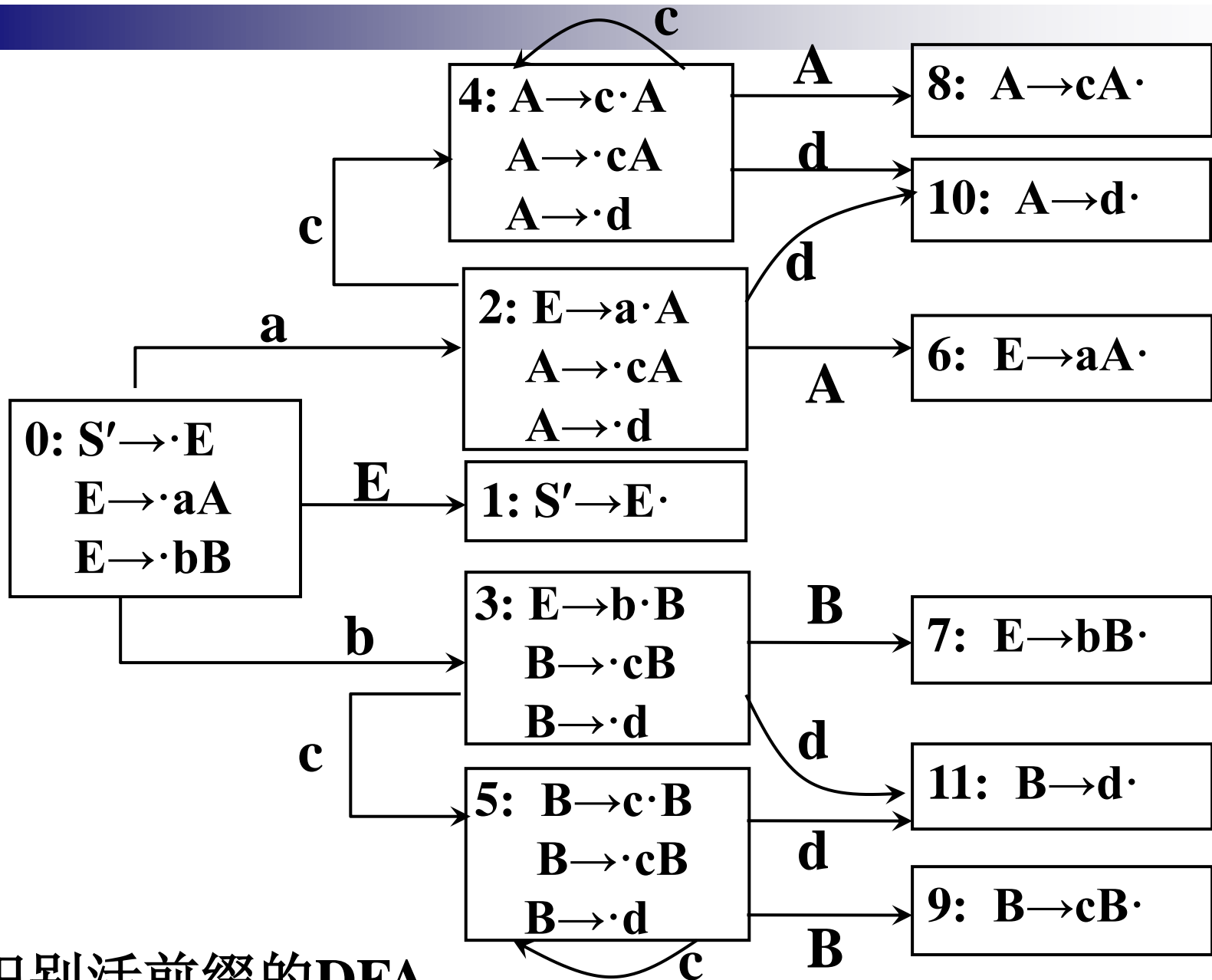
项目1为NFA的唯一初态，任何状态（项目）均认为是NFA的终态（活前缀识别态）

1. 若状态i为 $X \rightarrow X_1 \cdots X_{i-1} \cdot X_i \cdots X_n$ ，  
状态j为 $X \rightarrow X_1 \cdots X_{i-1} X_i \cdot X_{i+1} \cdots X_n$ ，  
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 $X \rightarrow \alpha \cdot A \beta$ ，A为非终结符，  
则从状态i画一条 $\varepsilon$ 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。

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



识别活前缀的NFA



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**。
  - $A \rightarrow \alpha \cdot$  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 $S' \rightarrow S$ 。

称G'是G的**拓广文法**。

- 拓广文法会有一个仅含项目 $S' \rightarrow S \cdot$ 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 $A \rightarrow \alpha \cdot B \beta$ 属于CLOSURE(I)，那么，对任何关于B的产生式 $B \rightarrow \gamma$ ，项目 $B \rightarrow \cdot \gamma$ 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。

# 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

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

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

例.文法 $G(S')$ :  $S' \rightarrow E$

$E \rightarrow aA | bB$

$A \rightarrow cA | d$

$B \rightarrow cB | d$

设 $I = \{S' \rightarrow \cdot E\}$

则 $CLOSURE(I) = \{S' \rightarrow \cdot E, E \rightarrow \cdot aA, E \rightarrow \cdot bB\}$

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

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

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

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

# LR(0)项目集规范族构造算法

PROCEDURE ITEMSETS( $G'$ );

BEGIN

$C := \{\text{CLOSURE}(\{S' \rightarrow \cdot S\})\};$

    REPEAT

        FOR  $C$ 中每个项目集 $I$ 和 $G'$ 的每个符号 $X$  DO

            IF  $\text{GO}(I, X)$ 非空且不属于 $C$  THEN

                把 $\text{GO}(I, X)$ 放入 $C$ 族中;

    UNTIL  $C$ 不再增大

END

- 转换函数 $\text{GO}$ 把项目集连接成一个DFA转换图.

例.文法 $G(S')$ :  $S' \rightarrow E$

$E \rightarrow aA | bB$

$A \rightarrow cA | d$

$B \rightarrow cB | d$

$C = \text{CLOSURE}(\{S' \rightarrow \cdot E\}) = \{\{S' \rightarrow \cdot E, E \rightarrow \cdot aA, E \rightarrow \cdot bB\}\} = I_0$

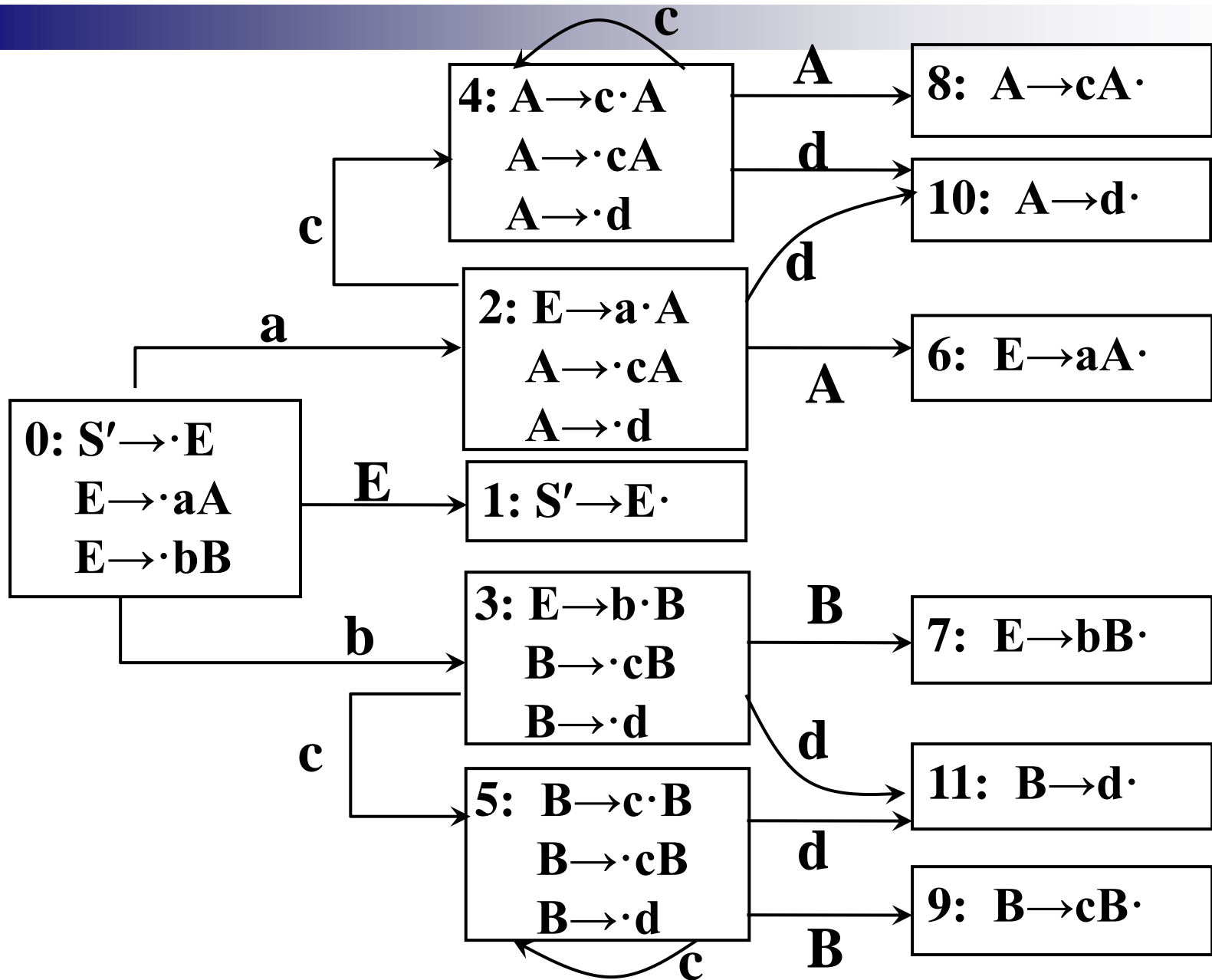
$GO(I_0, E) = \text{closure}(J) = \text{closure}(\{S' \rightarrow E \cdot\})$   
 $= \{S' \rightarrow E \cdot\} = I_1$

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

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

.....

可得12个项目集, 即为上面的DFA



# 有效项目

- 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)。

## ■ 文法G(S')

$S' \rightarrow E$

$E \rightarrow aA | bB$

$A \rightarrow cA | d$

$B \rightarrow cB | d$

## ■ 考虑活前缀: **bc** :

项目:  $B \rightarrow \mathbf{c.B}$     $B \rightarrow \mathbf{.cB}$     $B \rightarrow \mathbf{.d}$

$S' \Rightarrow E \Rightarrow \mathbf{bB} \Rightarrow \mathbf{bcB}$

$S' \Rightarrow E \Rightarrow \mathbf{bB} \Rightarrow \mathbf{bcB} \Rightarrow \mathbf{bccB}$

$S' \Rightarrow E \Rightarrow \mathbf{bB} \Rightarrow \mathbf{bcB} \Rightarrow \mathbf{bcd}$



# 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；

(2) 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')

(0)  $S' \rightarrow E$

(1)  $E \rightarrow aA$

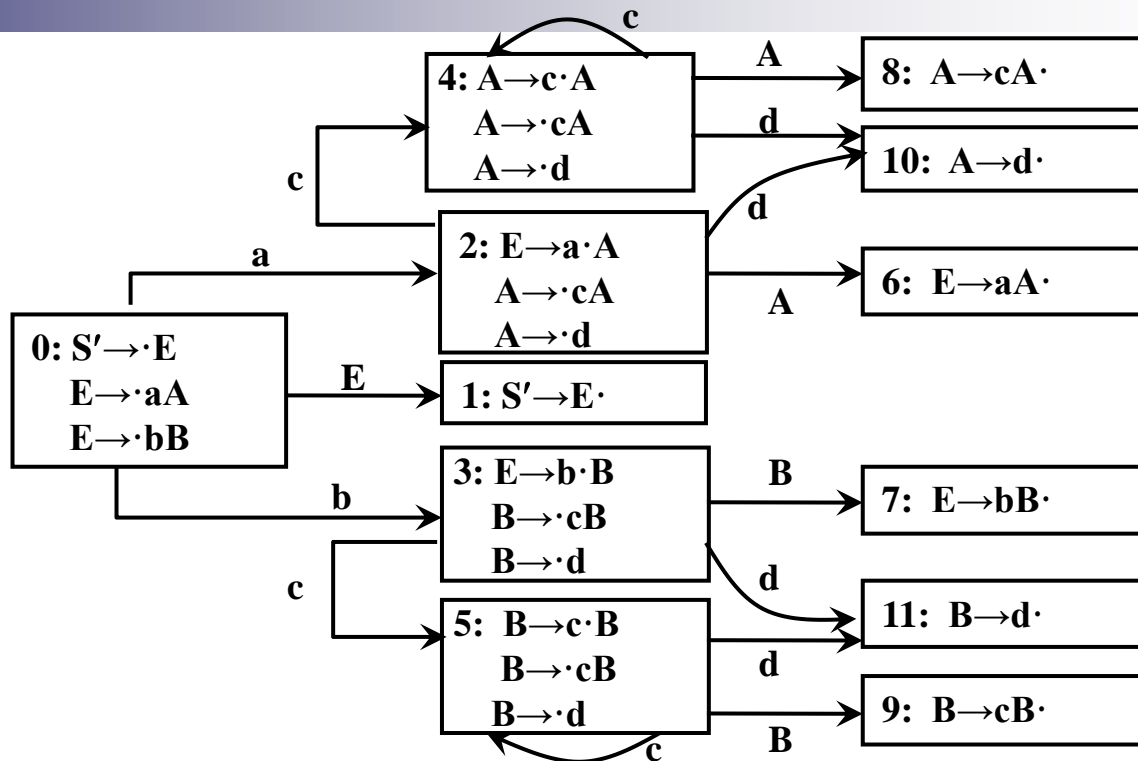
(2)  $E \rightarrow bB$

(3)  $A \rightarrow cA$

(4)  $A \rightarrow d$

(5)  $B \rightarrow cB$

(6)  $B \rightarrow d$



状态	ACTION					GOTO		
	a	b	c	d	#	E	A	B
0								
1								
2								
3								
4								
5								9
6								
7								
8								
9								
10								
11								

## ■ 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	#

# LR(0)文法的局限性

- LR(0)文法太简单，文法中很多情况不能通过LR(0)来解决
- 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：

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

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

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

$I_3: T \rightarrow F \cdot$

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

$I_5: F \rightarrow i \cdot$

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

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

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

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

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

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

# SLR冲突解决办法

- 假定一个LR(0)规范族中含有如下的一个项目集(状态) $I = \{X \rightarrow \alpha \cdot b \beta, A \rightarrow \alpha \cdot, B \rightarrow \alpha \cdot\}$ 。FOLLOW(A)和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集合有#), 则:

1. 若a是某个  $a_i$ ,  $i=1,2,\dots,m$ , 则移进;
2. 若  $a \in FOLLOW(B_i)$ ,  $i=1,2,\dots,n$ , 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。

# SLR(1)分析表构造方法

- 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”。

- 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。

# SLR(1) 分析表构造实例

- (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)  $F \rightarrow i$

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

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

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

$I_3: T \rightarrow F \cdot$

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

$I_5: F \rightarrow i \cdot$

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

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

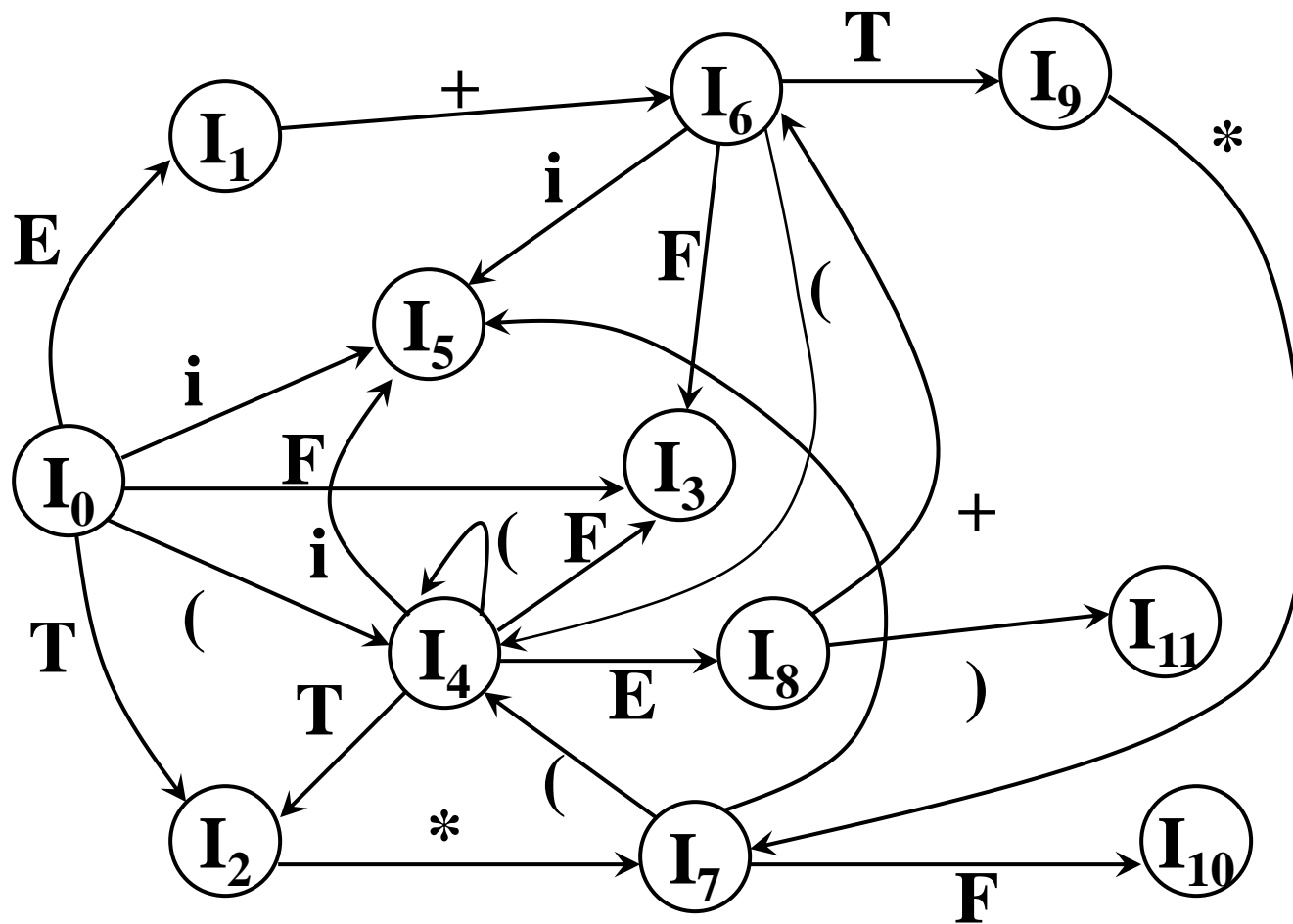
$I_8: F \rightarrow (E \cdot)$   
 $E \rightarrow E \cdot +T$

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

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

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

# SLR(1)分析表构造实例（续）



- $I_1$ 、 $I_2$ 和 $I_9$ 都含有“移进 - 归约”冲突。
- $\text{FOLLOW}(E) = \{\#, ), +\}$   
与移进项目 $\{ * \}$ 没有交集
- $\text{FOLLOW}(S') = \{ \# \}$   
与移进项目 $\{ + \}$ 没有交集

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

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

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

项目集规范族 $I_1, I_2, I_9$ 中看上去含有移进-规约冲突，但是可以通过不同的终结符号来区分到底执行移进动作还是规约动作，使SLR分析表依然可以保证不含多重入口。

。

由 $FOLLOW(E) = \{ \#, ), + \}$ ,  $FOLLOW(S') = \{ \# \}$ , 构造分析表:

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			



# SLR文法的局限性

## ■ 非SLR文法示例：

(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:

$I_0: S' \rightarrow \cdot S$   
 $S \rightarrow \cdot L = R$   
 $S \rightarrow \cdot R$   
 $L \rightarrow \cdot * R$   
 $L \rightarrow \cdot i$   
 $R \rightarrow \cdot L$

$I_1: S' \rightarrow S \cdot$

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

$I_3: S \rightarrow R \cdot$

$I_4: L \rightarrow * \cdot R$   
 $R \rightarrow \cdot L$   
 $L \rightarrow \cdot * R$   
 $L \rightarrow \cdot i$

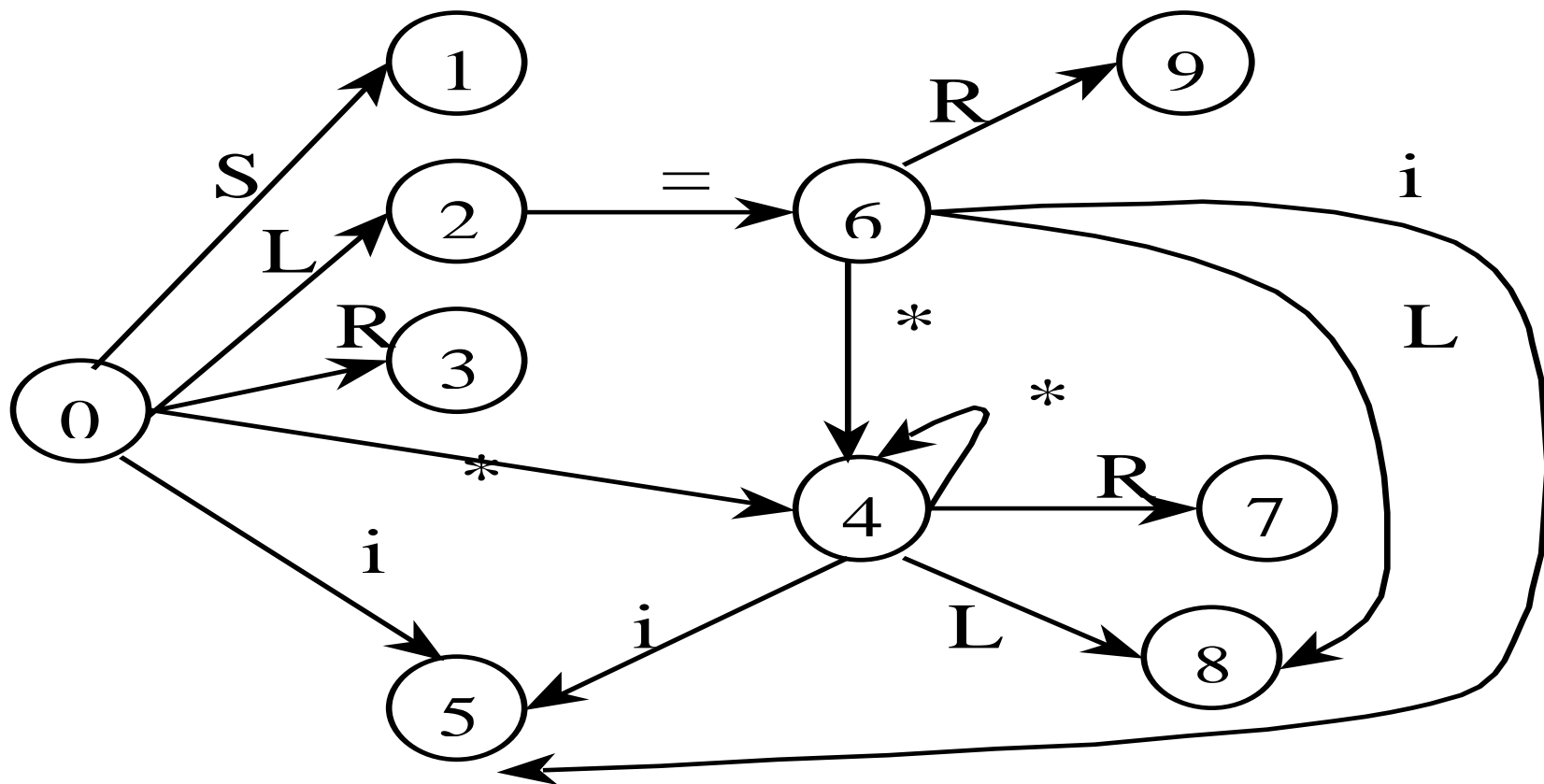
$I_5: L \rightarrow i \cdot$

$I_6: S \rightarrow L = \cdot R$   
 $R \rightarrow \cdot L$   
 $L \rightarrow \cdot * R$   
 $L \rightarrow \cdot i$

$I_7: L \rightarrow * R \cdot$

$I_8: R \rightarrow L \cdot$

$I_9: S \rightarrow L = R \cdot$



活前缀识别器

(0)  $S' \rightarrow S$

(1)  $S \rightarrow L=R$

(2)  $S \rightarrow R$

(3)  $L \rightarrow *R$

(4)  $L \rightarrow i$

(5)  $R \rightarrow L$

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

- $I_2$  含有“移进 - 归约”冲突。
- $FOLLOW(R) = \{#, =\}$ , 移进项目和规约项目都是=, SLR(1)无法解决

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。

(0)  $S' \rightarrow S$

(1)  $S \rightarrow L=R$

(2)  $S \rightarrow R$

(3)  $L \rightarrow *R$

(4)  $L \rightarrow i$

(5)  $R \rightarrow L$

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

实际上 “ $R=$ ”为前缀的规范句型不会出现  
而是含有 “ $*R=$ ”为前缀的规范句型  
**FOLLOW集合不能提供精确的展望信息**

- 在SLR方法中,如果项目集 $I_i$ 含项目 $A \rightarrow \alpha \cdot$ 而且下一输入符号 $a \in \text{FOLLOW}(A)$ ,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 $\alpha$ 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。

# 规范LR分析冲突解决方法

- 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** $[A \rightarrow \alpha \cdot \beta, a_1 a_2 \dots a_k]$ ， $\beta \neq \epsilon$ ，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 $\alpha$ 归约为A。

- 一般情况，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 $\gamma$ 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 $\omega$ 的第一个符号，或者 $a$ 为 $\#$ 而 $\omega$ 为 $\epsilon$ 。

- 为构造有效的LR(1)项目集族，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  $\gamma$  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

$$\begin{array}{c} * \\ S \Rightarrow \delta A a \chi \Rightarrow \delta \alpha B \beta a \chi \end{array} \quad \gamma = \delta \alpha$$

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

$$\begin{array}{c} * \\ \therefore \beta a \chi \Rightarrow b \omega \end{array}$$

若  $B \rightarrow \xi$  是产生式

$$\begin{array}{c} * \qquad * \\ \text{则 } S \Rightarrow \delta \alpha B \beta a \chi \Rightarrow \gamma B b \omega \Rightarrow \gamma \xi b \omega \end{array}$$

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

## ■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 $GO(I, X)$ 定义为：

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

其中

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

## ◆ 文法 $G'$ 的LR(1)项目集族 $C$ 的构造算法:

**BEGIN**

**$C := \{\text{CLOSURE}(\{[S' \rightarrow \cdot S, \#]\})\};$**

**REPEAT**

**FOR  $C$ 中每个项目集 $I$ 和 $G'$ 的每个符号 $X$  DO**

**IF  $\text{GO}(I, X)$ 非空且不属于 $C$ , THEN**

**把 $\text{GO}(I, X)$ 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“sj”
  2. 若项目 $[A \rightarrow \alpha \cdot, a]$ 属于 $I_k$ , 则置 $ACTION[k, a]$ 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 $GO(I_k, A) = I_j$ , 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，  
 $LR(0) \subset SLR \subset LR(1) \subset \text{无二义文法}$

## ■ 拓广文法 $G(S')$

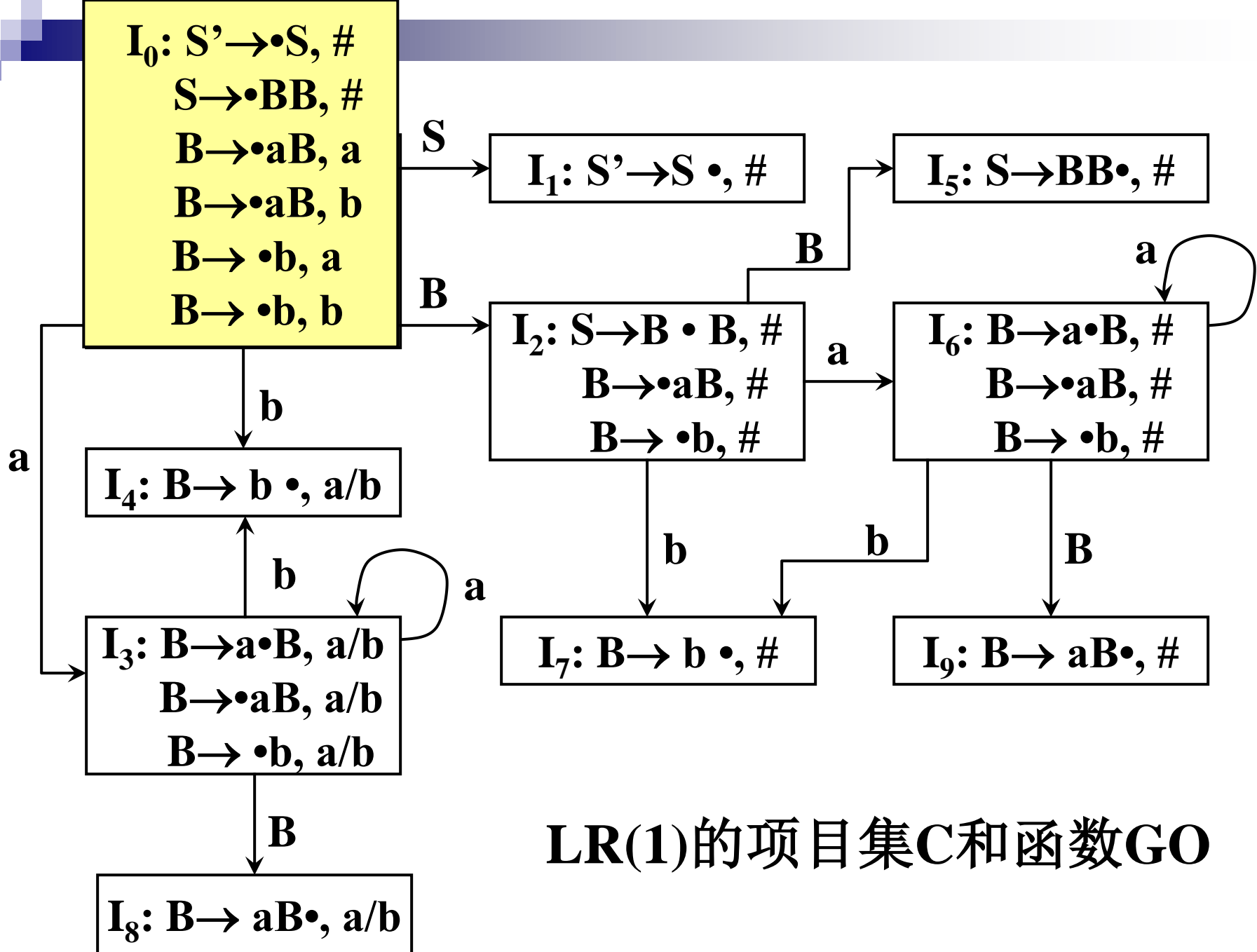
(0)  $S' \rightarrow S$

(1)  $S \rightarrow BB$

(2)  $B \rightarrow aB$

(3)  $B \rightarrow b$





LR(1)的项目集C和函数GO

# 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		

## ■ 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	#Bab	#
9	0269	#BaB	#
10	025	#BB	#
11	01	#S	#

acc

## ■ 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

# 语法分析自动产生工具——YACC

- **YACC——Yet Another Compiler- Compiler**
  - Stephen(1979) LALR(1)分析法
  - GNU Bison: 基本兼容YACC,可与Flex一起使用
  - Berkeley Yacc
- **The Lex&Yacc Page**
  - <http://dinosaur.compilertools.net>

# 练习

判断：

1. 由最右推导得到的句型称为规范句型。
2. 对于任何文法，都能将其改写成LL(1)文法。
3. 如果一个文法的任何产生式右部都不含有两个相继的非终结符，则该文法是算符优先文法。
4. 一个无二义文法的句型的句柄是不唯一的。
5. 在规范规约过程中，分析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。
6. 任何LR(1)文法都是SLR(1)文法。

## 练习

- 下面哪些文法是无二义文法？（ ）
  - A. LL(1)文法
  - B. 算符优先文法
  - C. SLR(1)文法
  - D. LR(1)文法
- 请给出LR(0)文法、SLR文法、LR(1)文法、无二义文法的包含关系。

## 练习

- 给定文法G(S):

$S \rightarrow (A) | a$

$A \rightarrow A + S | S$

求非终结符A的FIRSTVT集合。



## 练习

给定文法 $G(S)$ :

$$S \rightarrow (L) | aS | a$$
$$L \rightarrow L, S | S$$

求句型 $(S, (a))$ 的短语、直接短语和句柄。

## 练习

■ 对于某文法规范句型aBcDef，如果句柄是Bc，下面哪些字符是活前缀（ ）

- A.  $\varepsilon$
- B. Bc
- C. aB
- D. aBcD

## 练习

- 给定文法 $G(S')$ ，请构造该文法识别活前缀的DFA，并判定该文法是否为LR(0)文法、是否为SLR文法。

$S' \rightarrow S$

$S \rightarrow iSeS$

$S \rightarrow iS$

$S \rightarrow a$

## 思考

**Q: LR分析表、识别活前缀的DFA、项目集规范族之间的关系**

- LR分析表表示项目之间的转换关系
- 识别活前缀的DFA表明识别了哪一个句柄、识别了某一个终结符或非终结符后转向哪一个项目
- 项目集规范族表明现在的活前缀，按哪条产生式规约或下一个可能出现的字符。