



第五章 语法分析—— 自下而上分析

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 $S \xRightarrow{*}$ 输入串 α (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
- 算符优先分析方法
- 规范归约
- LR分析方法

例. 给定文法 **G**:

(1) $S \rightarrow aAcBe$

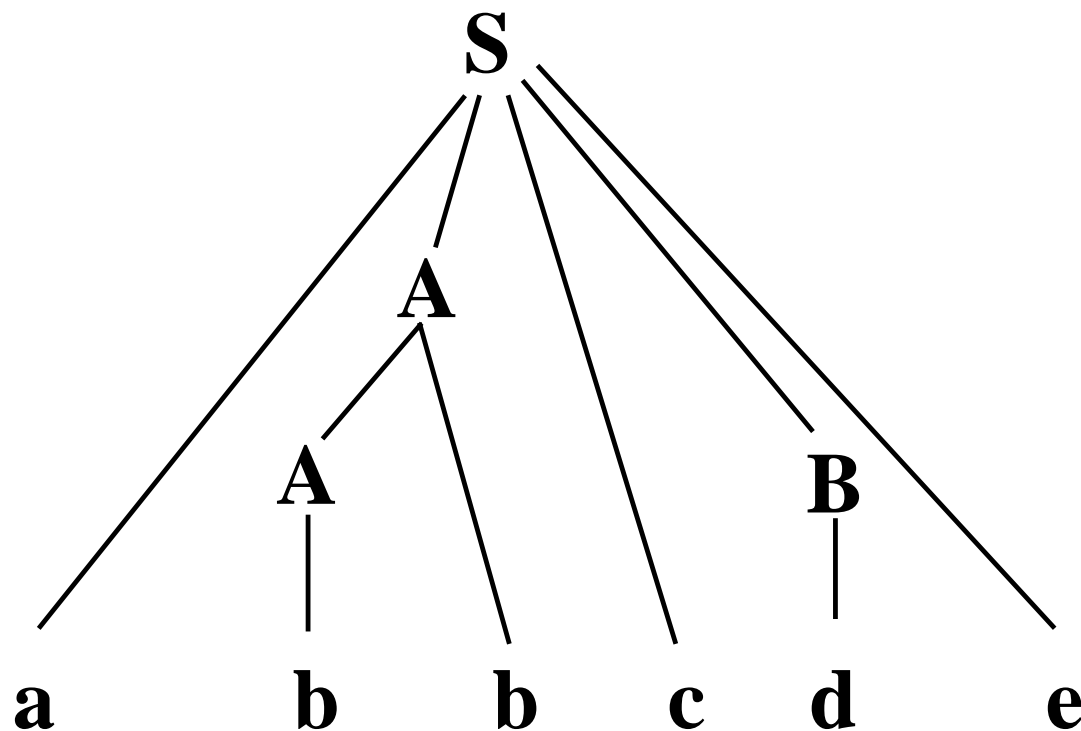
(2) $A \rightarrow b$

(3) $A \rightarrow Ab$

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

归约过程如下（分析树）：

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



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) **$S \rightarrow aAcBe$**

(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:

e
B
b
A
a

abbcde

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)

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

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, 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

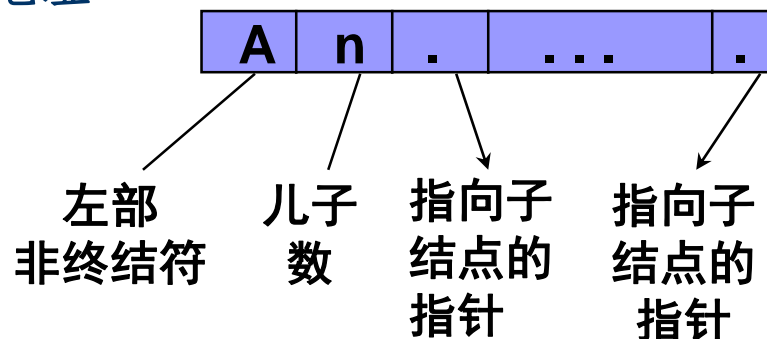
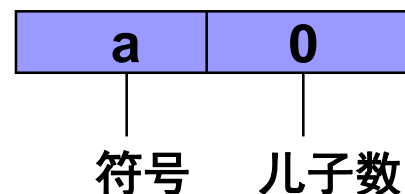
□ 移进符号 **a** 时, 构造表示端末结 **a** 的数据结构, 其地址与 **a** 同时进栈

□ 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

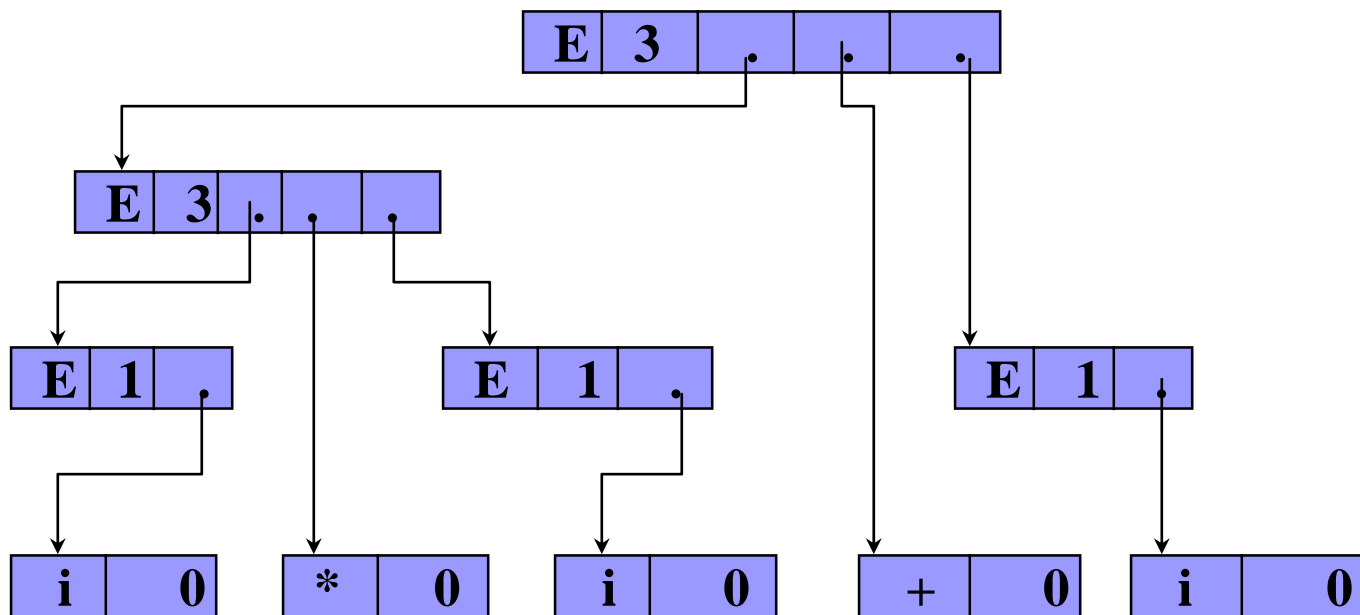
□ 接受时, 语法树已经构成, **S** 及根地址在栈中

符号	结点地址

符号栈



例. $i * i + i$ 的语法树



给定文法 **G**:

(1) $S \rightarrow aA$

(2) $C \rightarrow a|ab$

(3) $D \rightarrow ab$

(4) $A \rightarrow b$

(5) $B \rightarrow b$

输入串 **ab** 是否为句子?

自下而上分析的基本问题

$$X \rightarrow \alpha b \beta$$

$$A \rightarrow \alpha$$

$$B \rightarrow \alpha$$

- 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

■ 算符优先分析方法

■ 规范归约

■ 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）。
- 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：

$\square a < b$ a的优先级低于b

$\square a = b$ a的优先级等于b

$\square a > b$ 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：

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

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 ϵ -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): $P \rightarrow (E) \quad \therefore (E)$

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

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

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

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

由(4): $P \rightarrow (E), \quad E \Rightarrow E+T \quad \therefore (\leq +, + \geq)$

■ ■ ■

∴ **G**为算符优先文法

(# 看作终结符号，作为句子括号)

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 **$a \equiv b$** 的终结符对。

$a \equiv b$ 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) =

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

LASTVT (P) =

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

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**拆空为止。

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

$T \rightarrow T, S \mid S$

求 $FIRSTVT(S)$, $FIRSTVT(T)$

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

T	,
T	(
T	^
T	a

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

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

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

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

FIRSTVT主程序:

BEGIN

FOR 每个非终结符 P 和终结符 a DO

$F[P,a] := \text{FALSE};$

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

$F[P,a] := \text{true};$

把 (P, a) 下推进STACK栈

END;

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。

- 假定有个产生式的一个候选形为

...**aP**...

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

- 假定有个产生式的一个候选形为

...**Pb**...

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

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

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

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

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

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

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

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

约定任何终结符号有: $a > \#$, $\# < a$

短语

■ 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 β 是句型 $\alpha\beta\delta$ 相对于非终结符 A 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。

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

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

短语: $d, Ab, aAbcde$

句型语法树和句型的短语

■ 短语

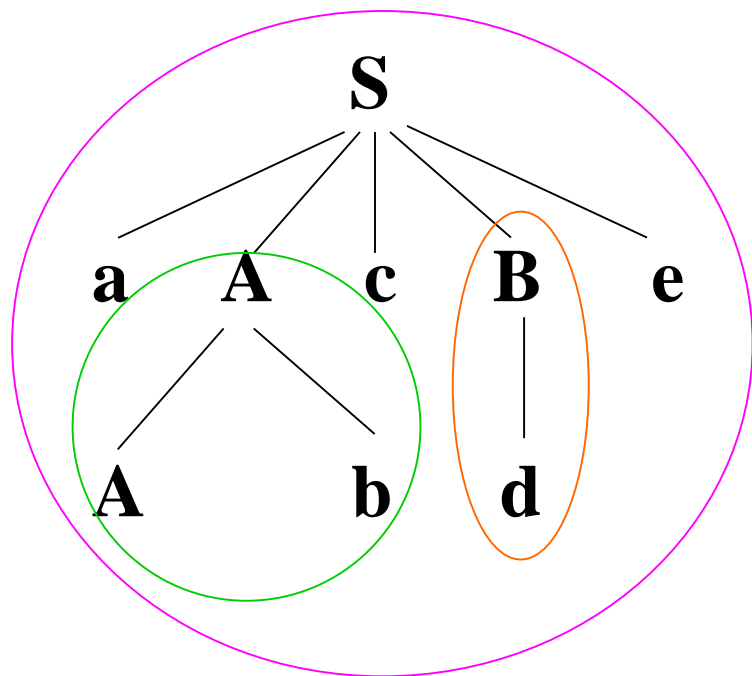
- 句型语法树中每棵子树（某个结点连同它的所有子孙组成的树）的所有叶子结点从左到右排列起来形成一个相对于子树根的短语。

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。

句型 $aAbcde$ 的语法树为：



短语： d , Ab , $aAbcde$

最左素短语

■ 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

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

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

■ 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) $E \rightarrow E+T \mid T$

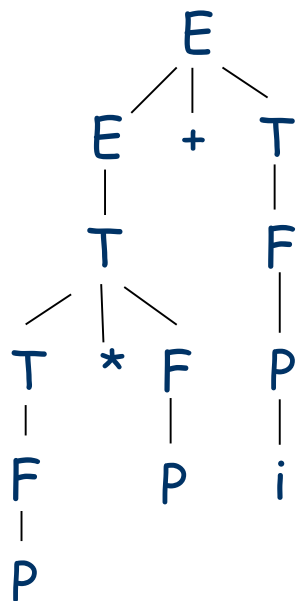
(2) $T \rightarrow T*F \mid F$

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

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

求句型 **$P*P+i$** 的短语、素短语、
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) $E \rightarrow E+T \mid T$

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

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

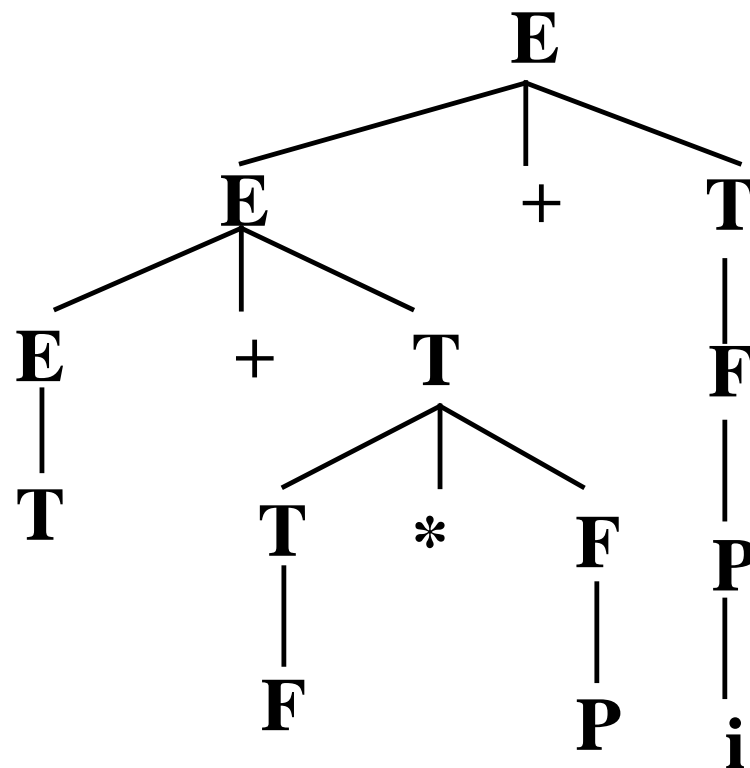
(4) $P \rightarrow (E) \mid i$

句型: $T+F*P+i$

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

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

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

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

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

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

最左素短语

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

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

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

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

$$\begin{array}{c} R \\ \# N_1 a_1 \dots a_{j-1} \overbrace{N_j a_j \dots N_i a_i N_{i+1}}^R a_{i+1} \dots N_n a_n N_{n+1} \# \\ \qquad \qquad \qquad < \qquad \qquad = \qquad \qquad > \end{array}$$

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

	+	*	\uparrow	()	i		+	*	\uparrow	()	i
E	T	T	T	T	F	T	E	T	T	T	F	T	T
T	F	T	T	T	F	T	T	F	T	T	F	T	T
F	F	F	T	T	F	T	F	F	F	T	F	T	T
P	F	F	F	T	F	T	P	F	F	F	F	T	T
FIRSTVT							LASTVT						

	+	*	\uparrow	()	i
+	>	<	<	<	>	<
*	>	>	<	<	>	<
\uparrow	>	>	<	<	>	<
(<	<	<	<	=	<
)	>	>	>		>	
i	>	>	>		>	

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: $T+F*P+i$

最左素短语: $F*P$

$+ < * > +$

	+	*	\uparrow	()	i
+	>	<	<	<	>	<
*	>	>	<	<	>	<
\uparrow	>	>	<	<	>	<
(<	<	<	<	=	<
)	>	>	>		>	
i	>	>	>		>	

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 **$<$** 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] := " # ";
2      REPEAT
3          把下一个输入字符读进;
4          IF S[k] ∈ VT THEN j := k;
5          WHILE S[j] > a DO
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	X_1	X_2	\dots	X_{k-j}
		\updownarrow	\updownarrow		\updownarrow
		$S[j+1]$	$S[j+2]$	\dots	$S[k]$

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

$i*(i+i)$

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

例. 对文法 **G**, 符号串 **$i*(i+i)$** 的分析过程如下:

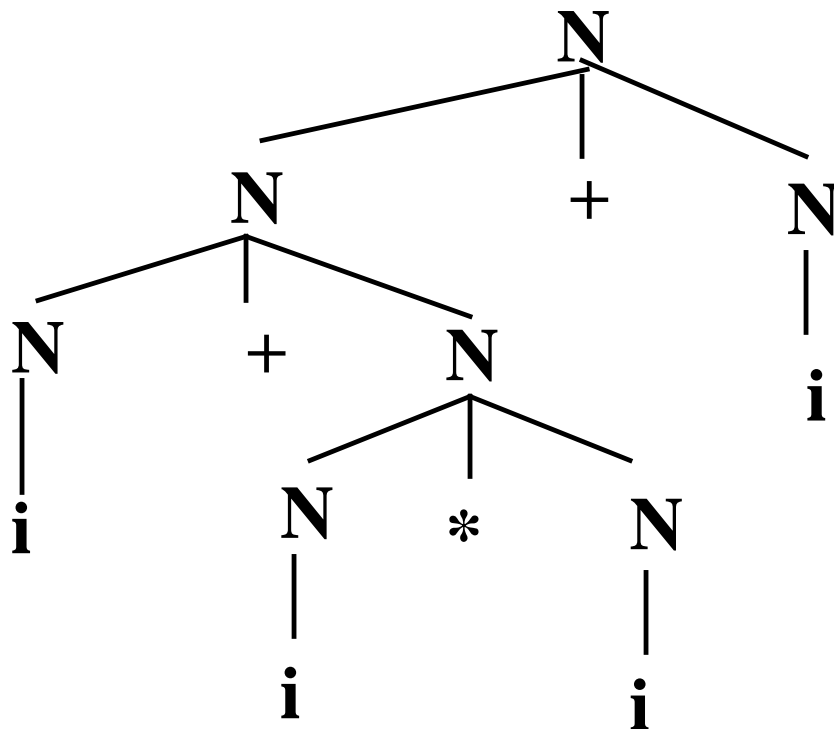
符号栈	关系	输入串	最左素短语
#	<.	i * (i+i) #	
# i	>.	* (i+i) #	i
#N	<.	* (i+i) #	
#N*	<.	(i+i) #	
#N*(<.	i +i) #	
#N*(i	>.	+i) #	i
#N*(N	<.	+ i) #	
#N*(N+	<.	i) #	
#N*(N+ i	>.) #	i
#N*(N+N	>.) #	N+N
#N*(N	=.) #	
#N*(N)	>.	#	(N)
# N*N	>.	#	N*N
#N	=.	#	
#N#		成功	

说明

- 约定任何终结符号有： $a > \#$ ， $\# < a$
- 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) $P \rightarrow (E) \mid i$

给出句子 $i + i * i + i$ 的算符优先分析的语法树



算符优先分析归约速度快，但容易误判。

内容线索

- ✓ 自下而上分析基本问题
- ✓ 算符优先分析方法
- 规范归约
- 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 β 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 β 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

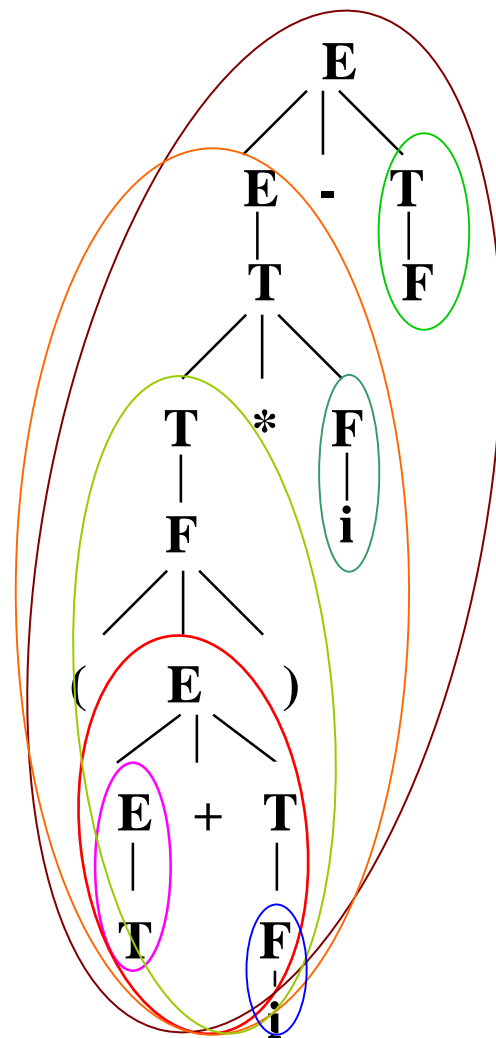
句型语法树和句型的短语、直接短语、句柄

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

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. 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: $T+F*P+i$

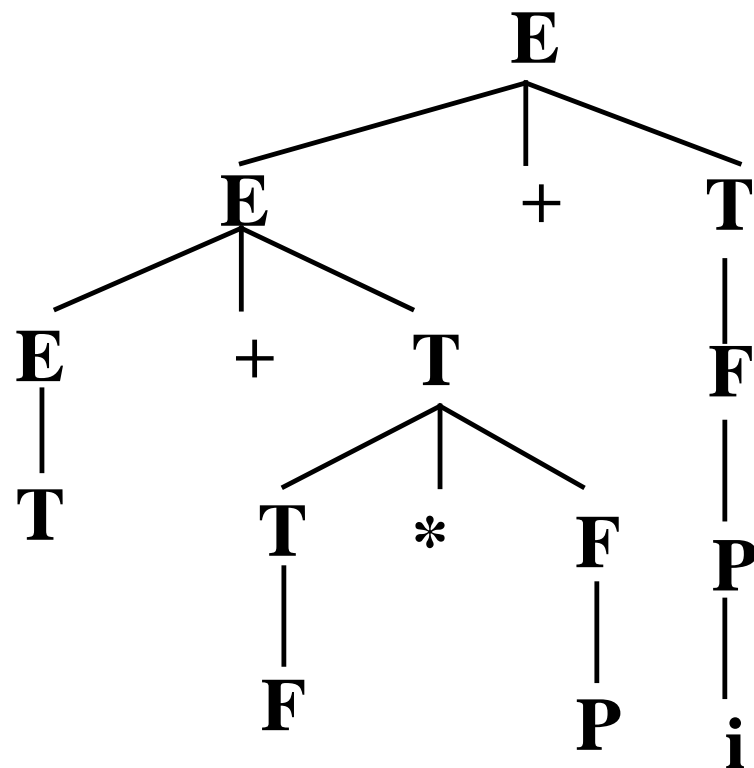
短语: $T, F, P, i, F*P,$
 $T+F*P, T+F*P+i$

直接短语: T, F, P, i

句柄: T

素短语: $F*P, i$

最左素短语: $F*P$



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 α 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 ， $0 < i \leq n$ ， α_{i-1} 是从 α_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。

规范归约是关于 α 的一个最右推导的逆过程

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

栈

↑

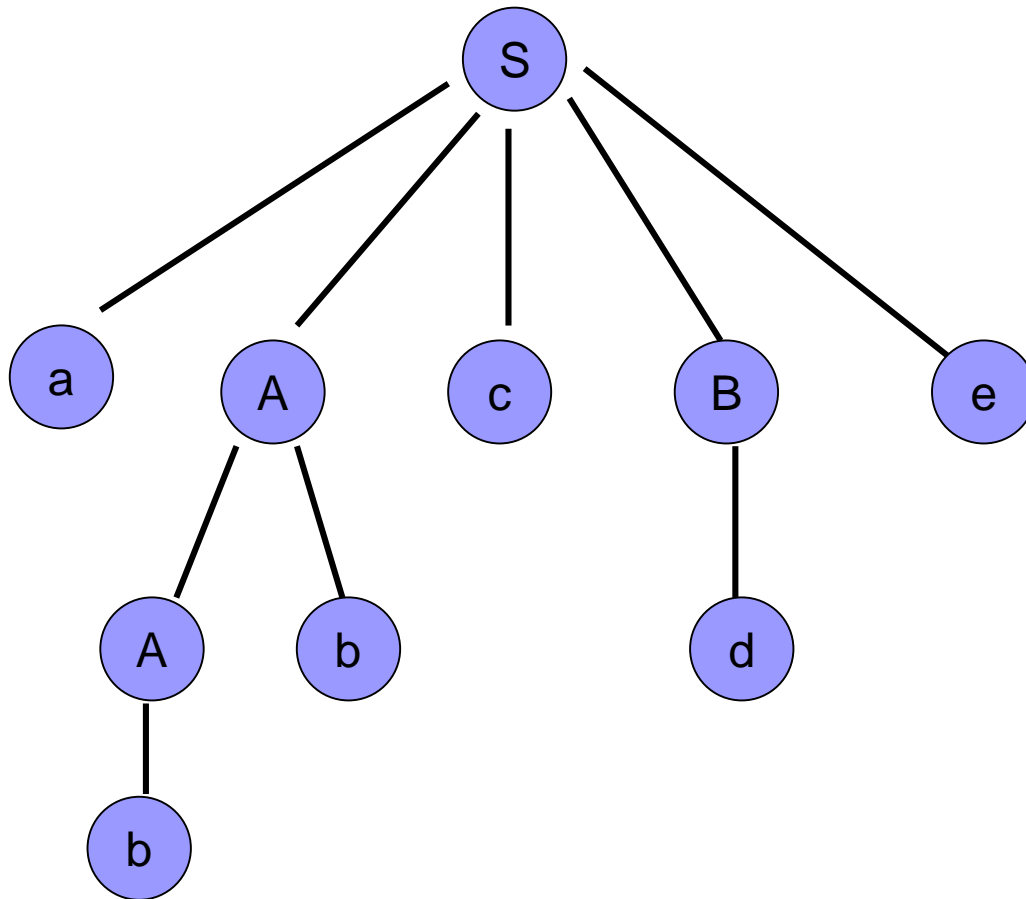
1	2	3	4	5	6	7	8	9	10
								e	
						$d \rightarrow B$		B	
			b		c	c	c	c	
	$b \rightarrow A$		$A \rightarrow A$		A	A	A	A	
a	a	a	a	a	a	a	a	a	S

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

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

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:

文法 **G (S)**

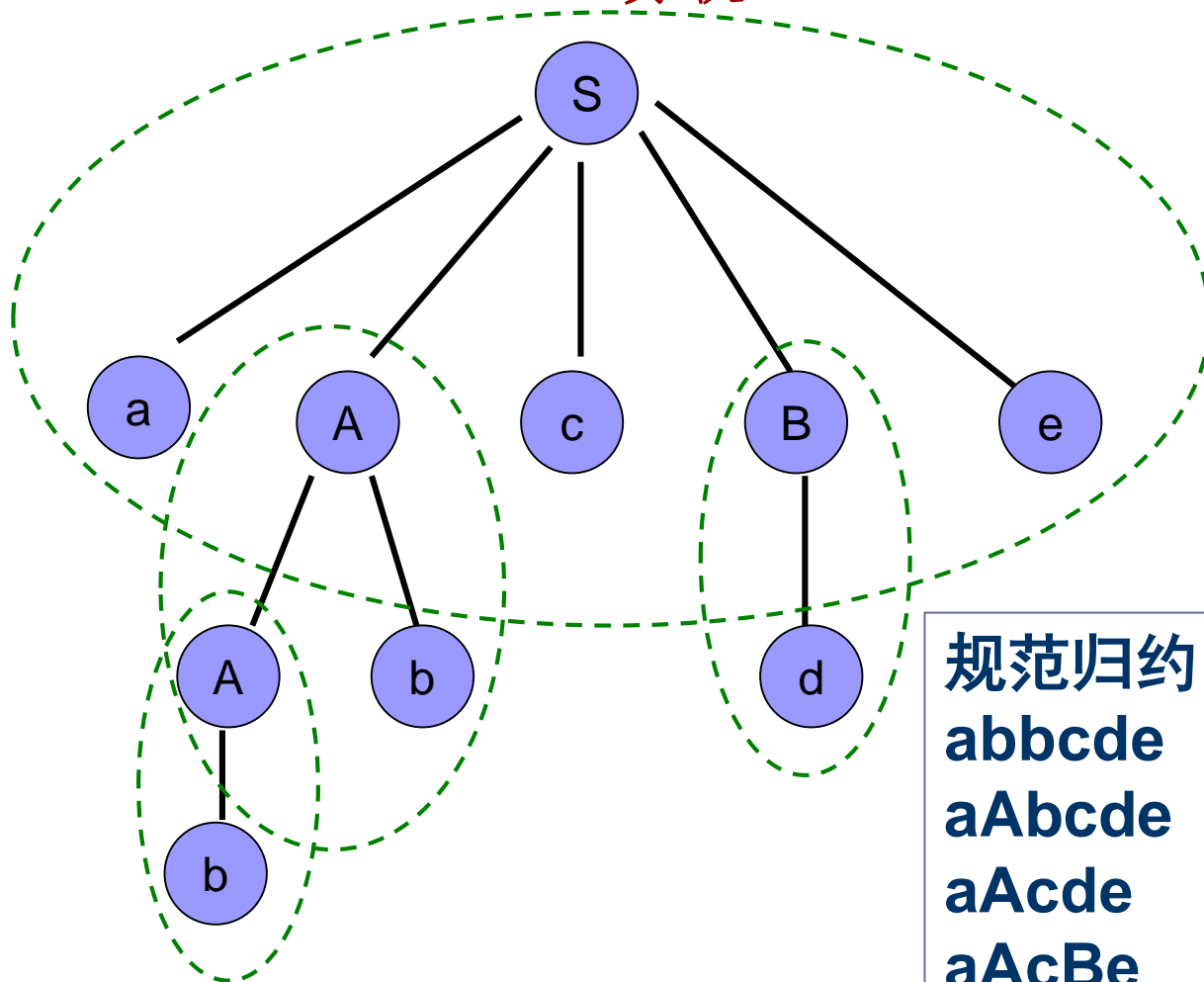
$S \rightarrow aAcBe$

$A \rightarrow b$

$A \rightarrow Ab$

$B \rightarrow d$

—— “**剪枝**”



规范归约

abbcde

aAbcde

aAcde

aAcBe

S

归约规则

$A \rightarrow b$

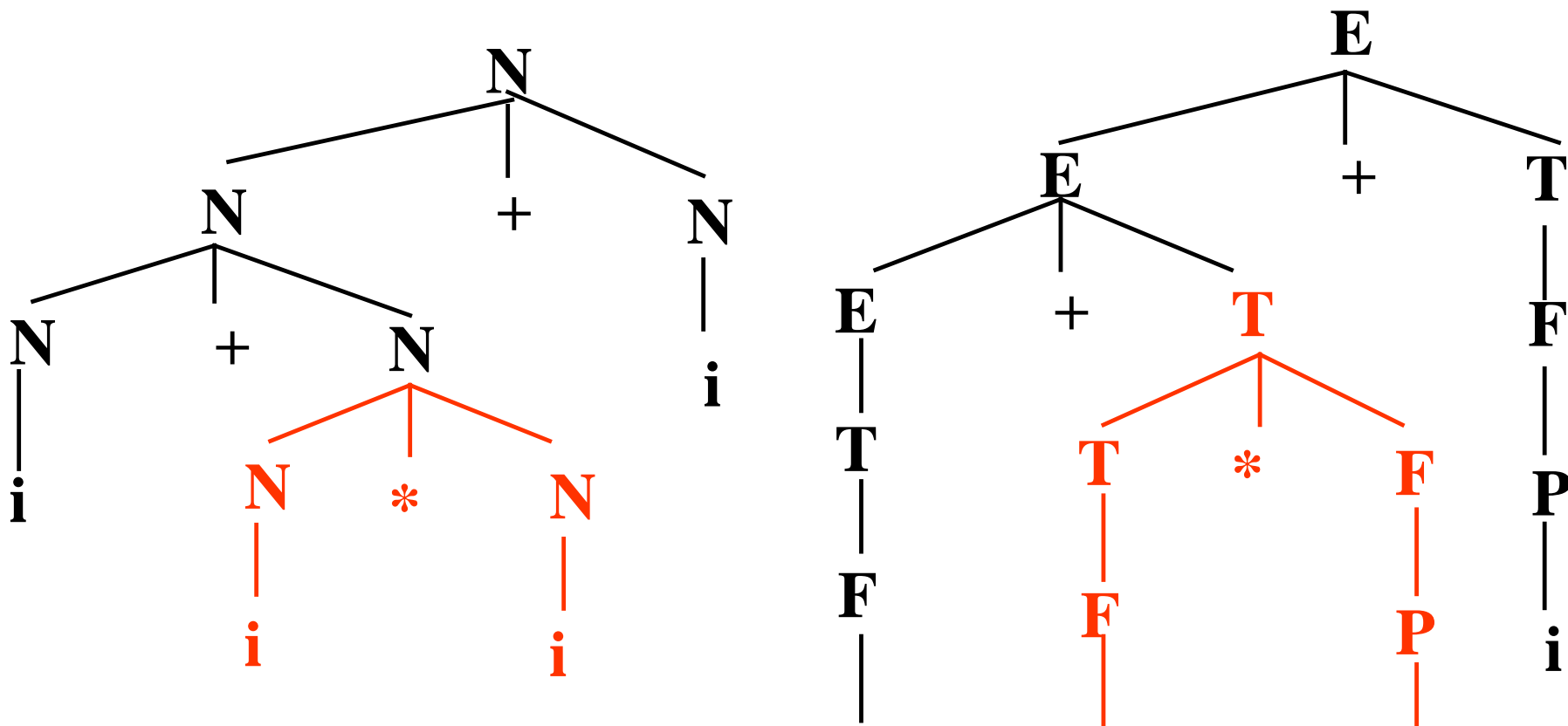
$A \rightarrow Ab$

$B \rightarrow d$

$S \rightarrow aAcBe$

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归约分析的语法树



算符优先分析相比规范归约, 其归约速度快, 但容易误判。

规范归约的基本问题

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

内容线索

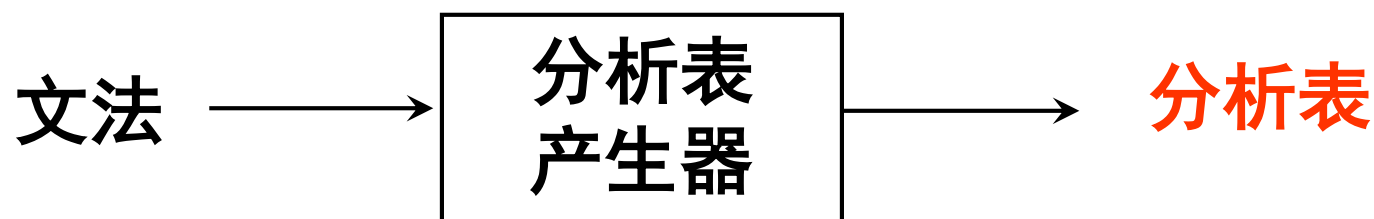
- ✓ 自下而上分析基本问题
- ✓ 算符优先分析方法
- ✓ 规范规约
- LR分析方法

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**规范归约**。
- 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



人物简介

■ Don E. Knuth高德纳(1938-)

- 编译程序: LR(k)
- 属性文法
- 算法: KMP算法
- 数字化排版TeX

■ 1974年图灵奖获得者

■ 《计算机程序设计艺术》作者



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

活前缀

- 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)。
 - 对于规范句型 $\alpha\beta\delta$, β 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。 (δ 必为终结字符串)

活前缀

- 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:

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

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

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

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。

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 $X \rightarrow X_1 \cdots X_{i-1} X_i \cdot X_{i+1} \cdots X_n$,

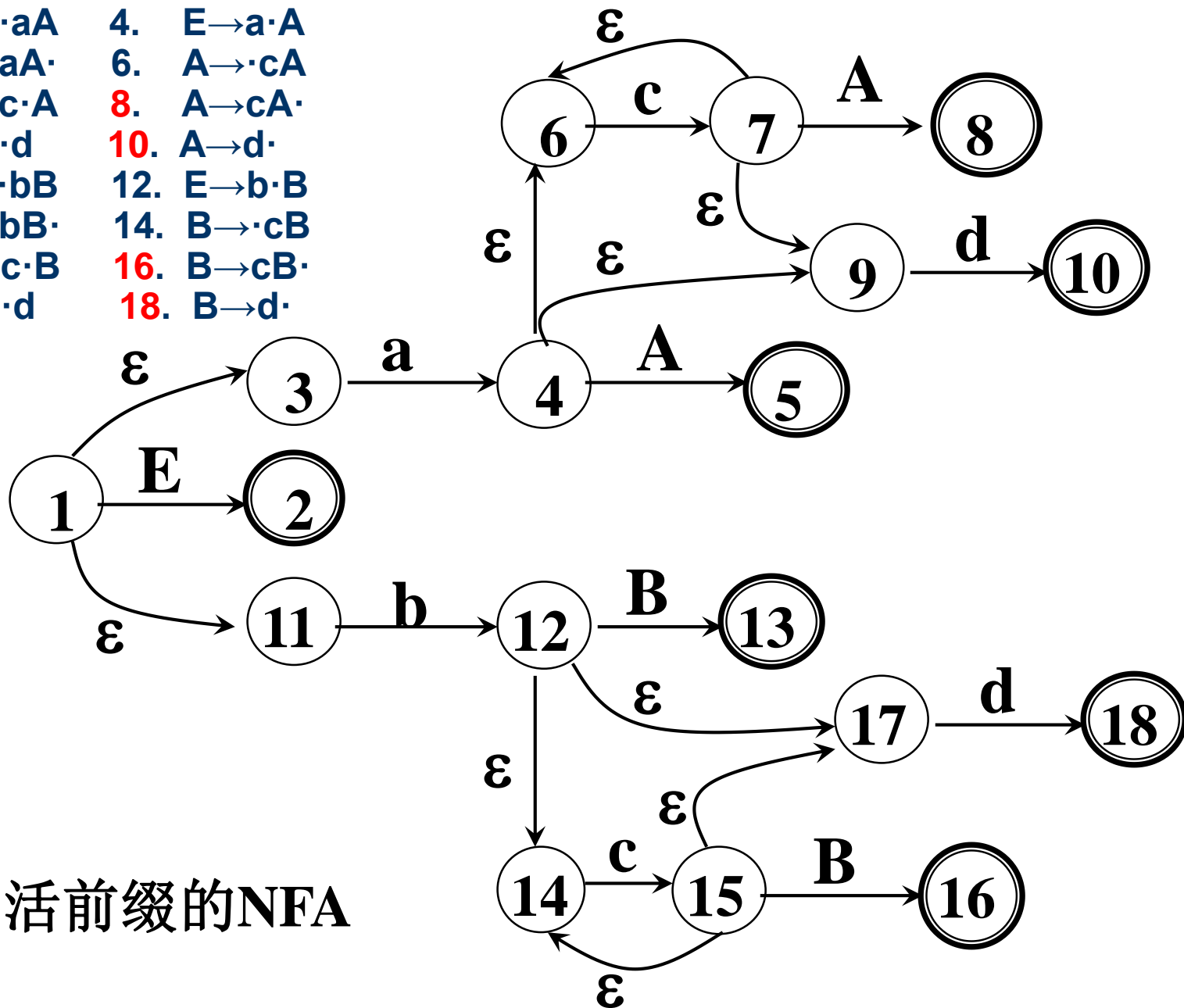
则从状态i画一条标志为 X_i 的有向边到状态j;

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

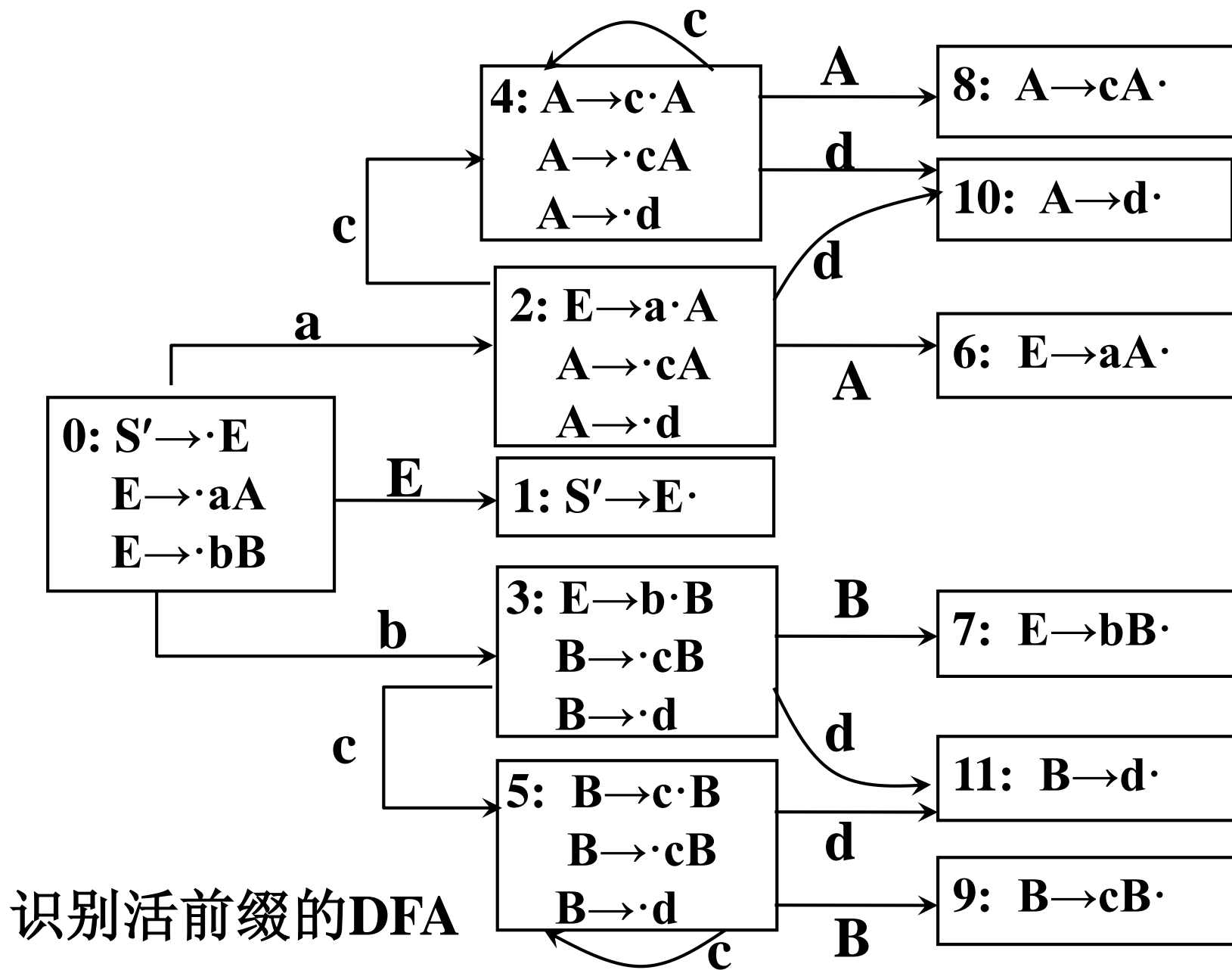
则从状态i画一条 ϵ 边到所有状态 $A \rightarrow \cdot \gamma$ 。

■ 把识别文法所有活前缀的NFA确定化。

- | | |
|-------------------------------|-------------------------------|
| 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



有效项目

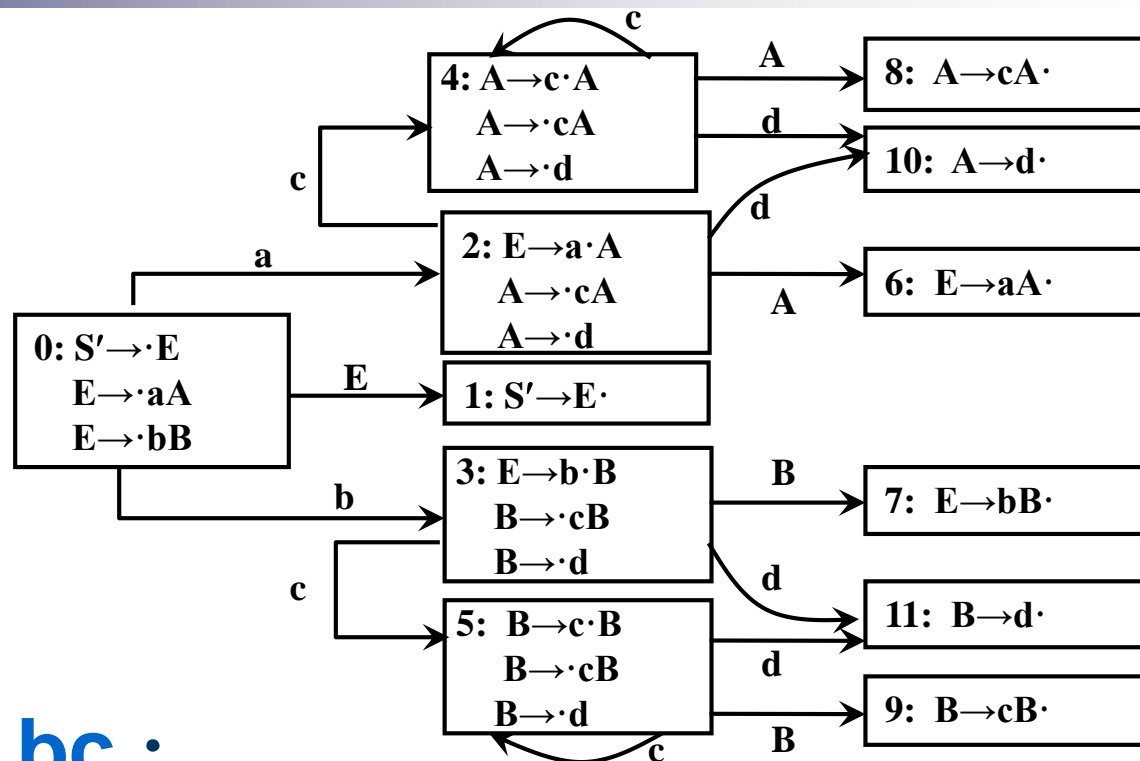
- 项目 $A \rightarrow \beta_1 \cdot \beta_2$ 对活前缀 $\alpha\beta_1$ 是有效的，其条件是存在规范推导

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

活前缀 $X_1 X_2 \dots X_m$ 的有效项目集从识别活前缀的 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)。

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。

■ $G(S')$
 $S' \rightarrow E$
 $E \rightarrow aA | bB$
 $A \rightarrow cA | d$
 $B \rightarrow cB | d$



■ 考虑活前缀: **bc** :

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

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

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

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

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".

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 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 $G(S')$: $S' \rightarrow E$

$E \rightarrow aA | bB$

$A \rightarrow cA | d$

$B \rightarrow cB | d$

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**GO**把项目集连接成一个**DFA**转换图.

例.文法**G(S')**: $S' \rightarrow E$

$E \rightarrow aA | bB$

$A \rightarrow cA | d$

$B \rightarrow cB | d$

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

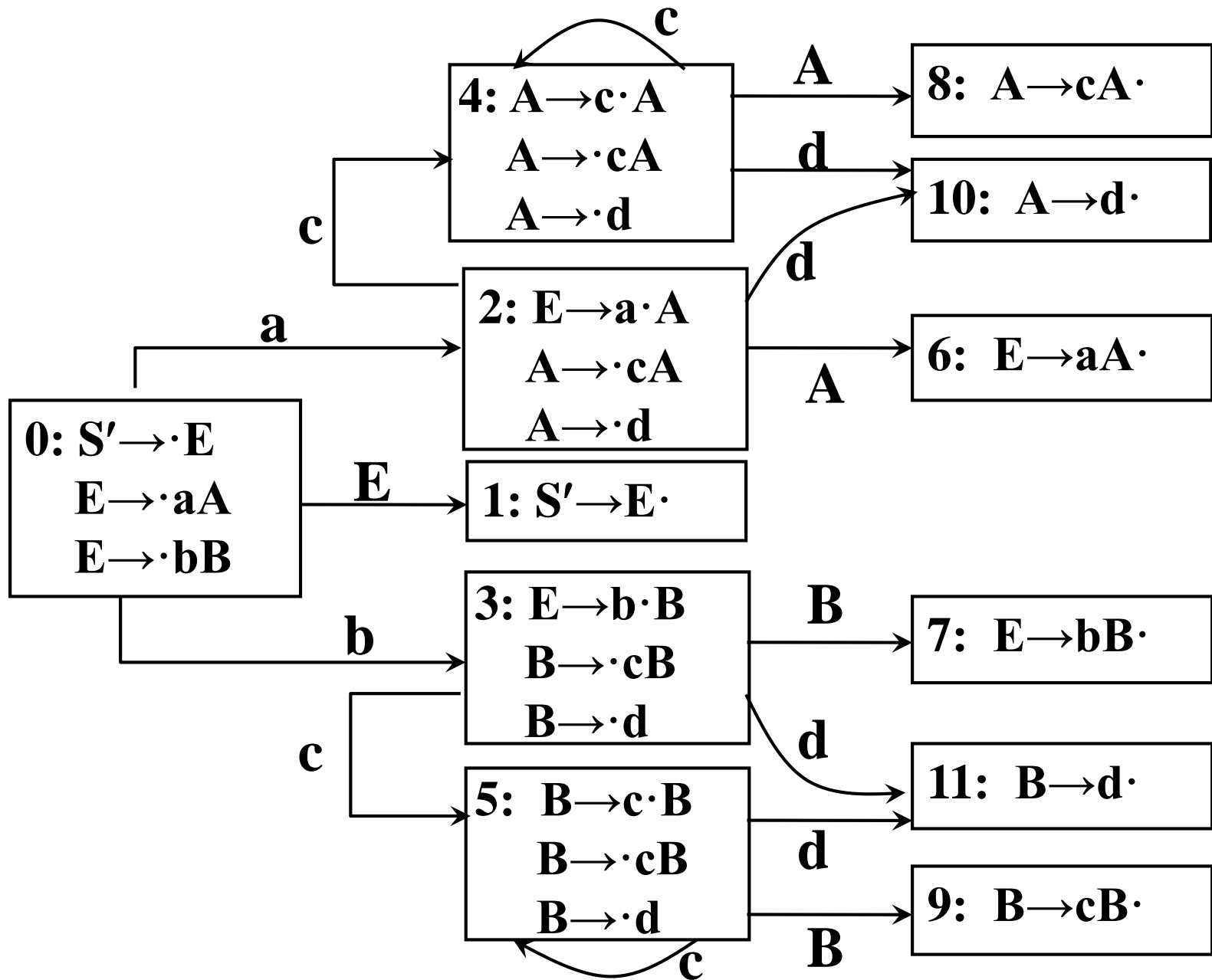
$GO(I_0, E) = \text{closure}(J) = \text{closure}(\{S' \rightarrow E \cdot\})$
 $= \{S' \rightarrow E \cdot\} = I_1$

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

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

.....

可得**12**个项目集,即为上面的**DFA**



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 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**

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) $E \rightarrow aA$

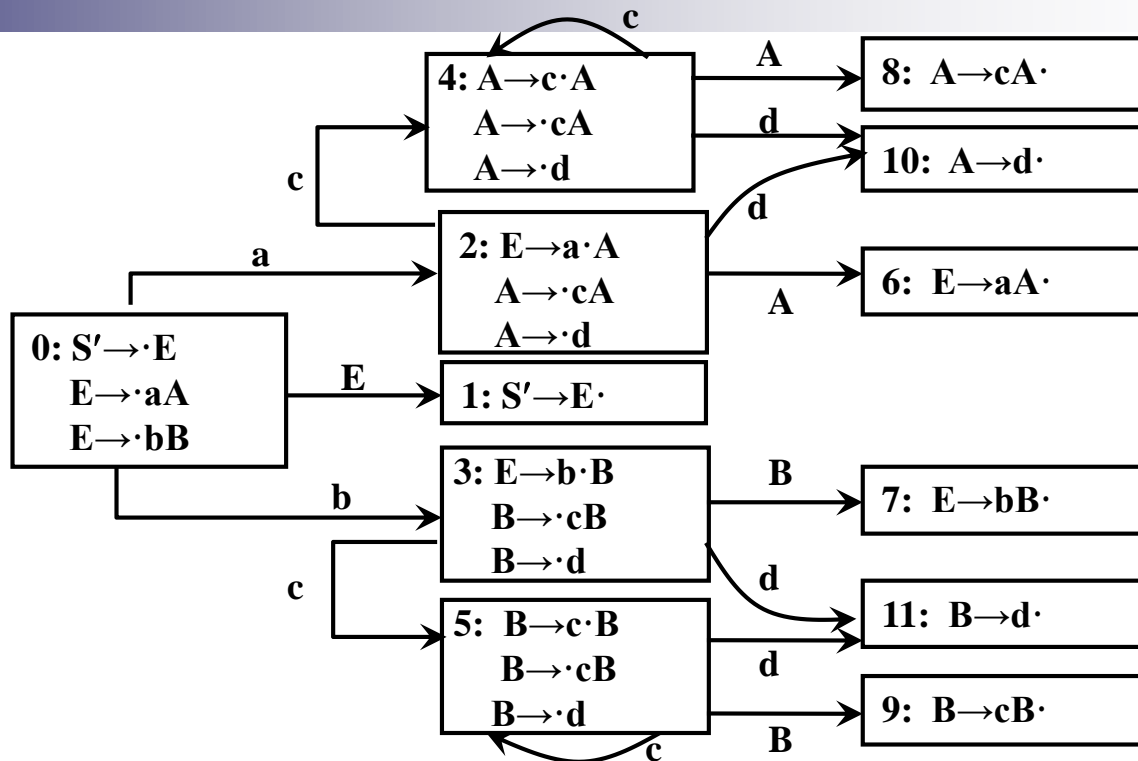
(2) $E \rightarrow bB$

(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								

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：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 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。

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。

SLR(1)分析表的构造

- 只有当一个文法**G**是**LR(0)**文法，才能构造出**LR(0)**分析表；
- 由于大多数适用的程序设计语言的文法不能满足**LR(0)**文法的条件，因此本节将介绍对于**LR(0)**规范族中冲突的项目集（状态）用向前查看一个符号的办法进行处理，以解决冲突。

SLR(1)分析表的构造

- 假定一个LR(0)规范族中含有如下的项目集（状态）I:

$$I = \{ X \rightarrow \alpha \cdot b\beta,$$

$$A \rightarrow \alpha \cdot ,$$

$$B \rightarrow \alpha \cdot \}$$

- 该项目集中含有移进-归约冲突和归约-归约冲突。

■ 如何解决这种冲突？

- **LR (0)** 在归约时不向前看输入符号；
- 在**LR (0)**基础上，如果存在移进-归约冲突或归约-归约冲突，则**LR (k)**方法通过向前看**k**个输入符号来解决冲突（利用上下文信息来消除当前的歧义）
- 因为只对有冲突的状态才向前查看一个符号，以确定做哪种动作，因而称这种分析方法为简单的**LR(1)分析法**，用**SLR(1)**表示

- 对于归约项目 $A \rightarrow \alpha \cdot$, $B \rightarrow \alpha \cdot$ 分别求 $\text{Follow}(A)$ 和 $\text{Follow}(B)$, 如果满足如下条件
 - $\text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \phi$
 - $\text{FOLLOW}(A) \cap \{b\} = \phi$
 - $\text{FOLLOW}(B) \cap \{b\} = \phi$
- 那么, 当在状态 **I** 时面临某输入符号为 **a** 时, 则构造分析表时用以下方法即可解决冲突动作。
 - (1) 若 $a=b$, 则移进。
 - (2) 若 $a \in \text{Follow}(A)$, 则用 $A \rightarrow \alpha$ 进行归约。
 - (3) 若 $a \in \text{Follow}(B)$, 则用 $B \rightarrow \alpha$ 进行归约。
 - (4) 此外, 报错。

SLR(1) 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对于形如 $I = \{ X \rightarrow \alpha \cdot b\beta, A \rightarrow \alpha \cdot, B \rightarrow \alpha \cdot \}$ 含有移进-归约冲突和归约-归约冲突的状态（项目集）满足下述情况：

- $\text{FOLLOW}(A) \cap \text{FOLLOW}(B) = \phi$

- $\text{FOLLOW}(A) \cap \{b\} = \phi$

- $\text{FOLLOW}(B) \cap \{b\} = \phi$

■ 则称 **G** 是一个 **SLR(1)** 文法。

SLR(1)分析表的构造

- 例如文法：
 - (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) $F \rightarrow i$

状态描述序列如下:

状态	项目集	后继符号	后继状态
I_0	$\{ S' \rightarrow \cdot E$ $E \rightarrow \cdot E + T$ $E \rightarrow \cdot T$ $T \rightarrow \cdot T * F$ $T \rightarrow \cdot F$ $F \rightarrow \cdot (E)$ $F \rightarrow \cdot i \}$	E E T T F $($ i	S_1 S_1 S_2 S_2 S_3 S_4 S_5
I_1	$\{ S' \rightarrow E \cdot$ $E \rightarrow E \cdot + T \}$	$\# S' \rightarrow E$ $+$	S_{12} S_6
I_2	$\{ E \rightarrow T \cdot$ $T \rightarrow T \cdot * F \}$	$\# E \rightarrow T \cdot$ $*$	S_{12} S_7
I_3	$\{ T \rightarrow F \cdot \}$	$\# T \rightarrow F$	S_{12}

状态	项目集	后继符号	后继状态
I_4	$\{F \rightarrow (\cdot E)$ $E \rightarrow \cdot E + T$ $E \rightarrow \cdot T$ $T \rightarrow \cdot T * F$ $T \rightarrow \cdot F$ $F \rightarrow \cdot (E)$ $F \rightarrow \cdot i \}$	E E T T F $($ i	S_8 S_8 S_2 S_2 S_3 S_4 S_5
I_5	$\{F \rightarrow i \cdot\}$	$\#F \rightarrow i$	S_{12}
I_6	$\{E \rightarrow E + \cdot T$ $T \rightarrow \cdot T * F$ $T \rightarrow \cdot F$ $F \rightarrow \cdot (E)$ $F \rightarrow \cdot i \}$	T T F $($ i	S_9 S_9 S_3 S_4 S_5
I_7	$\{T \rightarrow T * \cdot F$ $F \rightarrow \cdot (E)$ $F \rightarrow \cdot i \}$	F $($ i	S_{10} S_4 S_5

状态	项目集	后继符号	后继状态
I_8	$\{F \rightarrow (E \cdot)$ $E \rightarrow E \cdot + T\}$) +	S_{11} S_6
I_9	$\{E \rightarrow E + T \cdot$ $T \rightarrow T \cdot * F\}$	$\#E \rightarrow E + T$ *	S_{12} S_7
I_{10}	$\{T \rightarrow T * F \cdot\}$	$\#T \rightarrow T * F$	S_{12}
I_{11}	$\{F \rightarrow (E) \cdot\}$	$\#F \rightarrow (E)$	S_{12}
I_{12}	{ }		

由上图可见， I_1 、 I_2 和 I_9 的项目集均不相容，其有移进项目和归约项目并存，构造LR(0)分析表如下：

[illegible]

SLR (1) 方法

- 从上表也可见在 S_1, S_2, S_9 中存在移进-归约冲突。这个表达式不是LR(0)文法，也就不能构造LR(0)分析表，现在分别考查这三个项目（状态）中的冲突是否能用SLR(1)方法解决。
- 对于S1: $\{S' \rightarrow E \cdot, E \rightarrow E \cdot + T\}$
 - 由于 $\text{Follow}(S') = \{ \# \}$ ，而 $S' \rightarrow E \cdot$ 是唯一的接受项目，所以当且仅当遇到句子的结束符“#”号时才被接受。又因 $\{ \# \} \cap \{ + \} = \emptyset$ ，因此 S_1 中的冲突可解决。

SLR (1) 方法

■ 对于 S_2 : $S_2 = \{ E \rightarrow T \cdot, T \rightarrow T \cdot * F \}$

□ 计算 $\text{Follow}(E) = \{ \#, +,) \}$

□ 所以 $\text{Follow}(E) \cap \{ * \} = \emptyset$

□ 因此面临输入符为 ‘+’, ‘)’ 或 ‘#’ 号时, 则用产生式 $E \rightarrow T$ 进行归约。

□ 当面临输入符为 ‘*’ 号时, 则移进, 其它情况则报错。

SLR (1) 方法

- 对于 S_9 : $S_9 = \{ E \rightarrow E+T \cdot, T \rightarrow T \cdot *F \}$
 - 计算 $\text{Follow}(E) = \{ \#, +,) \}$
 - 所以 $\text{Follow}(E) \cap \{ * \} = \phi$
 - 因此面临输入符为 ‘+’, ‘)’ 或 ‘#’ 号时, 则用产生式 $E \rightarrow E+T$ 进行归约。
 - 当面临输入符为 ‘*’ 号时, 则移进。其它情况则报错。

总结 (1)

- 由以上考查, 该文法在 S_1 , S_2 , S_9 三个项目集(状态)中存在的移进-归约冲突都可以用 **SLR(1)** 方法解决, 因此该文法是 **SLR(1)** 文法。我们可构造其相应的 **SLR(1)** 分析表。
- **SLR(1)** 分析表的构造与 **LR(0)** 分析表的构造类似, 仅在含有冲突的项目集中分别进行处理。

总结 (2)

- 进一步分析我们可以发现如下事实：如在状态 S_3 中，只有一个归约项目 $T \rightarrow F \cdot$ ，按照 **SLR(1)** 方法，在该项目中没有冲突，所以保持原来 **LR(0)** 的处理方法，不论当前面临的输入符号是什么都将用产生式 $T \rightarrow F$ 进行归约。
- 但很显然 T 的 Follow 集合不含 ‘(’ 符号，如果当前面临输入符是 ‘(’，也进行归约是错误的。

总结 (3)

- 因此可以对所有归约项目都采取SLR(1)的思想，即对所有非终结符都求出其Follow集合，这样凡是归约项目仅对面临输入符号包含在该归约项目左部非终结符的Follow集合中，才采取用该产生式归约的动作。
- 对于这样构造的SLR(1)分析表，称它为改进的SLR(1)分析表。

改进的SLR(1)分析表的构造方法

(1) 对于 $A \rightarrow \alpha \cdot X \beta$, $GO[S_i, X] \in S_j$, 若

$X \in V_T$, 则置 $actoin[S_i, X] = S_j$

$X \in V_N$, 则置 $goto[S_i, X] = j$

(2) 对于归约项目 $A \rightarrow \alpha \cdot \in S_i$, 若 $A \rightarrow \alpha$ 为文法的第 j 个产生式, 则对任何输入符号 a , 若 $a \in Follow(A)$, 则置 $action[S_i, a] = r_j$

(3) 若 $S \rightarrow \alpha \cdot \in S_i$, 则置 $action[S_i, \#] = acc$

(4) 其它情况均置出错。

状态	ACTION						GOTO		
	i	+	*	()	#	E	T	F
S ₀	S ₅			S ₄			1	2	3
S ₁		S ₆				acc			
S ₂		r ₂	S ₇		r ₂	r ₂			
S ₃		r ₄	r ₄		r ₄	r ₄			
S ₄	S ₅			S ₄			8	2	3
S ₅		r ₆	r ₆		r ₆	r ₆			
S ₆	S ₅			S ₄				9	3
S ₇	S ₅			S ₄					10
S ₈		S ₆			S ₁₁				
S ₉		r ₁	S ₇		r ₁	r ₁			
S ₁₀		r ₃	r ₃		r ₃	r ₃			
S ₁₁		r ₅	r ₅		r ₅	r ₅			

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

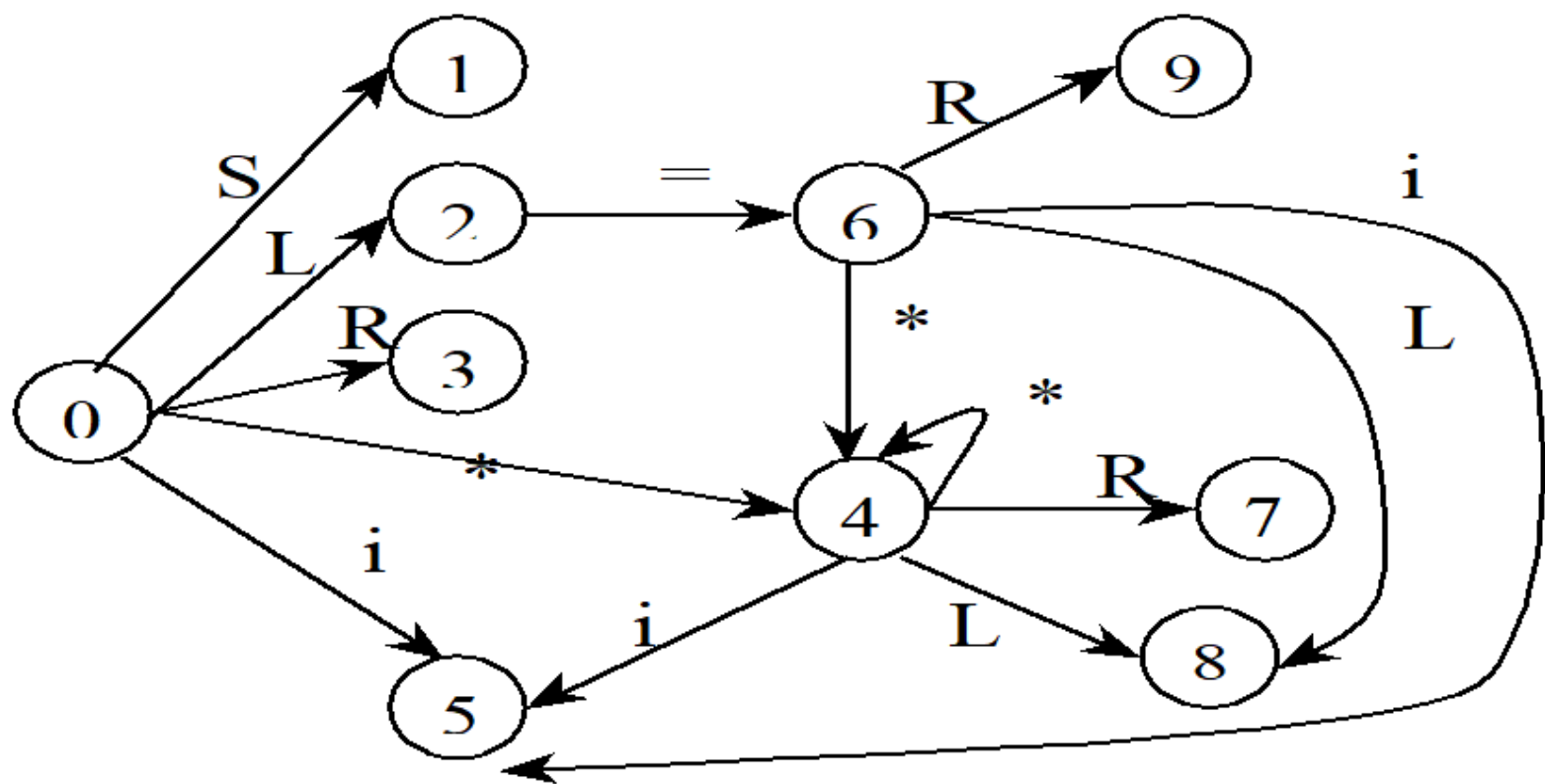
I_5 : $L \rightarrow i \cdot$

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 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。

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 α 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 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，需要两个函数CLOSURE和GO。

■ 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

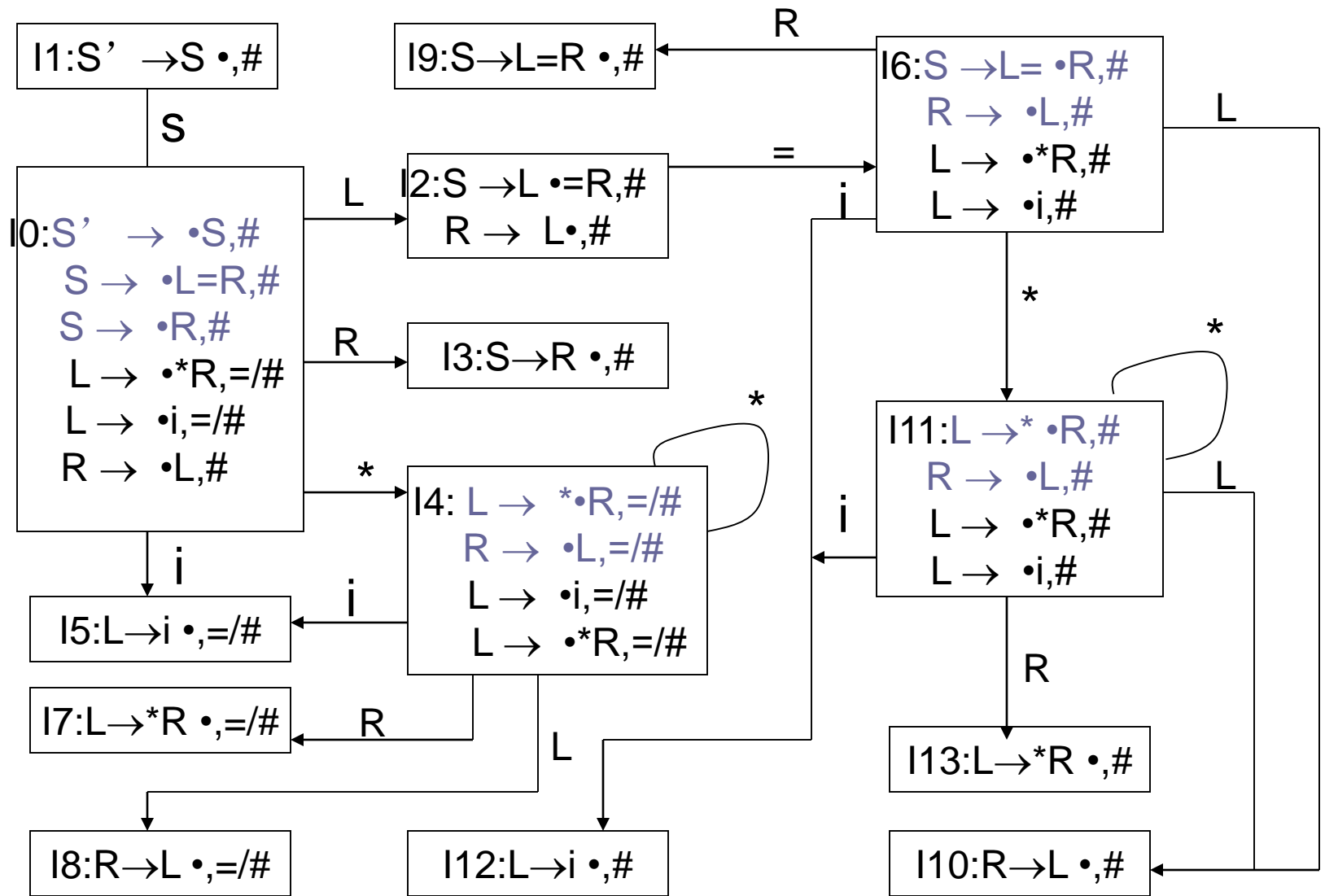
■ 构造LR(1)分析表的算法。

- 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，**
 $LR(0) \subset SLR \subset LR(1) \subset$ 无二义文法



前例的 LR(1)项目集及转换函数

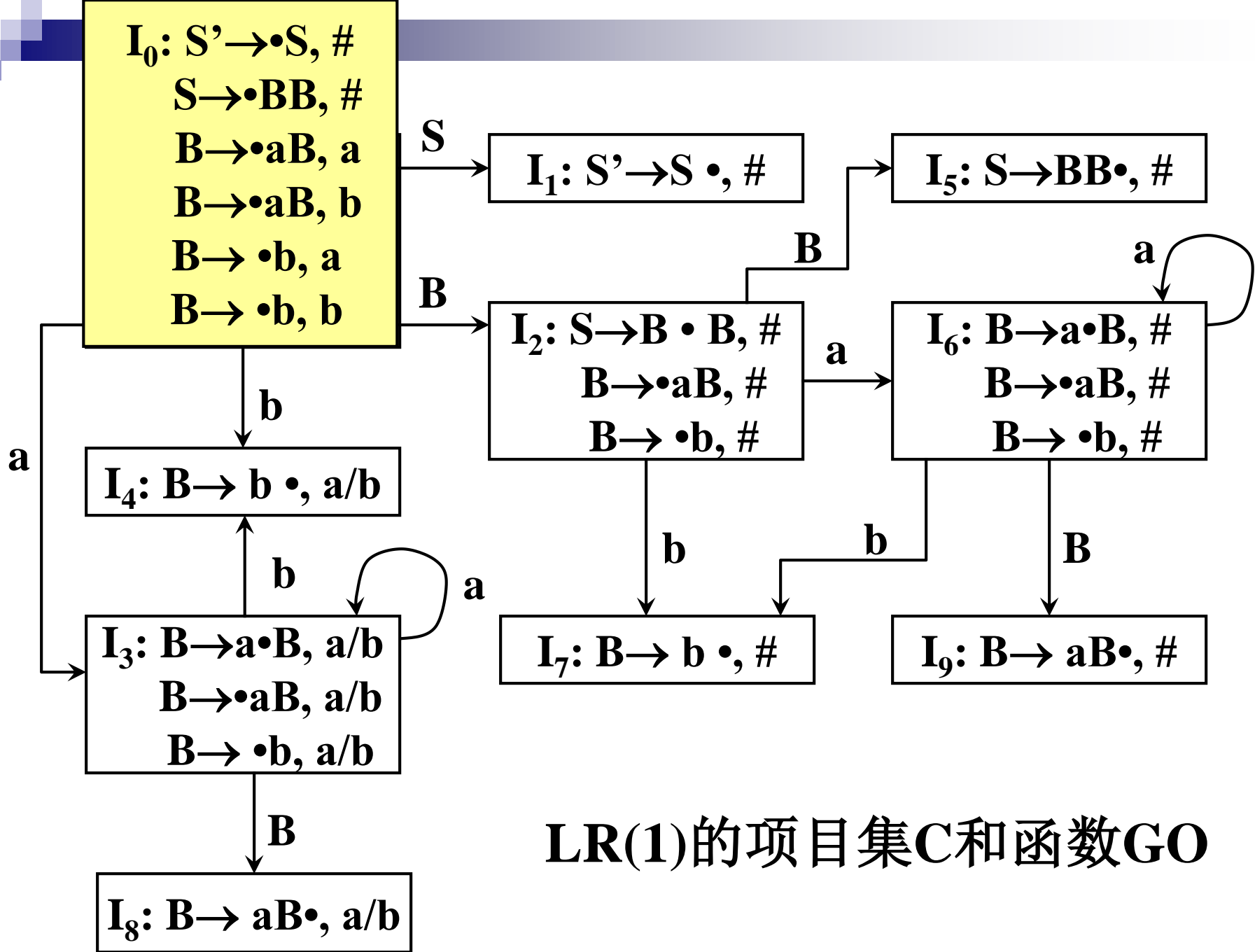
■ 拓广文法 $G(S')$

(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>#</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

作业

■ P133

☐ 1

☐ 2(2)

☐ 3(1)(2)(4)

☐ 5 (1)(2)(3)