同济大学 TONGJI UNIVERSITY

编译原理

第五章　语法分析——自下而上分析

丁志军

dingzj@tongji.edu.cn

概述

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

自上而下分析法	开始符号S $\overset{*}{\Rightarrow}$ 输入串α （推导）
自下而上分析法	输入串α $\overset{*}{\Rightarrow}$ 开始符号S （归约）

内容线索

1. 自下而上分析基本问题
2. 算符优先分析方法
3. 规范归约
4. LR分析方法

示例

给定文法 G：

(1) S→aAcBe

(2) A→b

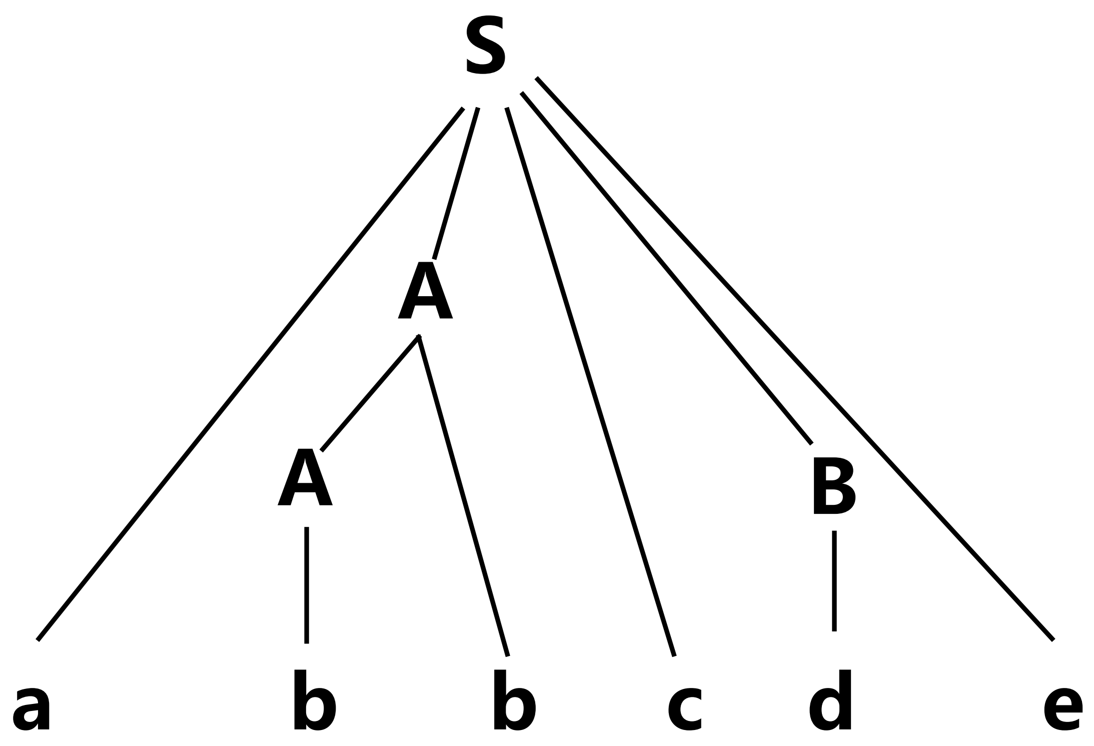
(3) A→Ab

(4) B→d

输入串 abbcde是否为句子?

归约过程如下（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。

示例

给定文法 G：

(1) S→aAcBe

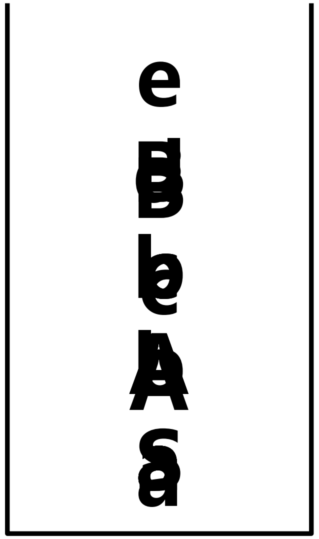
(2) A→b

(3) A→Ab

(4) B→d

输入串 abbcde是否为句子？

归约过程如下：



示例

给定文法 G：

(1) S→aAcBe

(2) A→b

(3) A→Ab

(4) B→d

输入串 abbcde是否为句子?

归约过程如下：

步骤：	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
动作：	进 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

符号栈的使用

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

	栈	输入串
初始	#	w#
最终	#S	#

示例

G: $E \to E+E \mid E*E \mid (E) \mid i$ 给出 i1*i2+i3 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$E \to i$
3	#E*	$i_2+i_3\#$	移进
4	$\#E*i_2$	$+i_3\#$	移进
5	#E*E	$+i_3\#$	归约$E \to i$
6	#E	$+i_3\#$	归约$E \to E*E$
7	#E+	$i_3\#$	移进
8	$\#E+i_3$	#	移进
9	#E+E	#	归约$E \to i$
10	#E	#	归约$E \to E+E$
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

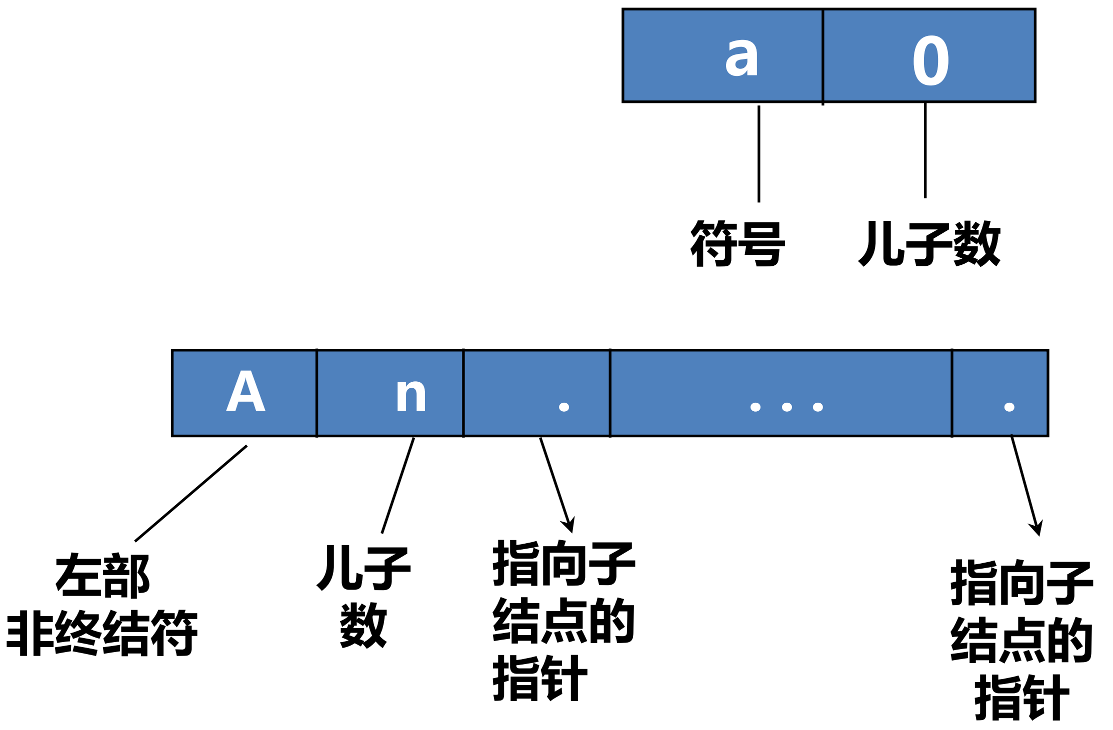
注: 可归约的串在栈顶,不会在内部

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 $A \rightarrow X_1X_2 \ldots X_n$ 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

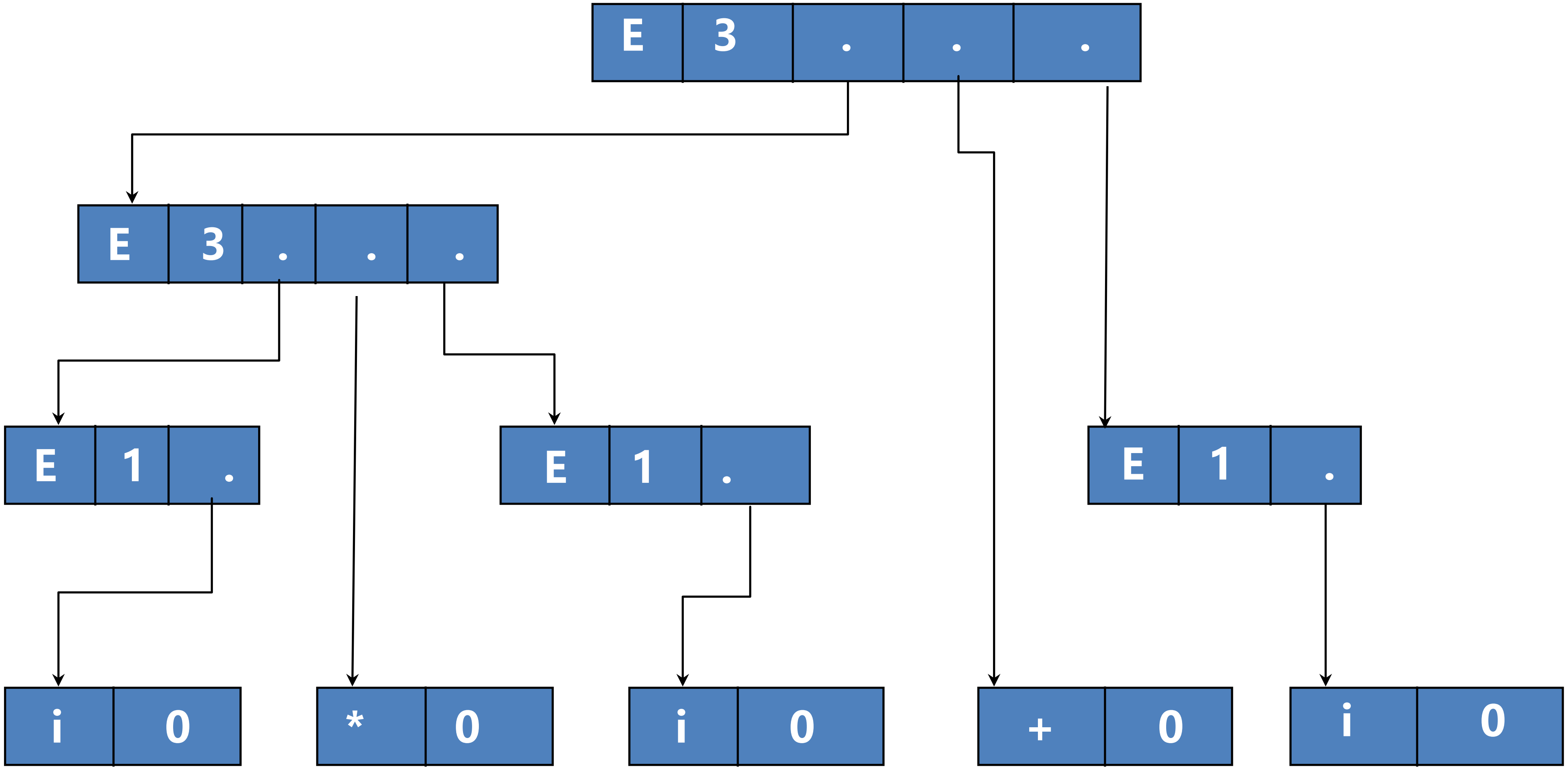
符号	结点地址

符号栈



示例

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

2. 算符优先分析方法

3. 规范归约

4. 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**：

$a \lessdot b$	a的优先级**低于**b
$a \doteq b$	a的优先级**等于**b
$a \gtrdot b$	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：

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

则称该文法为算符文法。

算符优先关系

- 设 G为算符文法且不含ε- 产生式， a, b∈ V_T, 算符间的优先关系定义为：

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

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

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

算符优先文法

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

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

则称 G 为**算符优先文法**。

示例

给定文法G：E→E+E|E*E|(E)|i　其中：V_T={+, *, i, (,)}。

G是算符文法

G是算符优先文法吗?

考察终结符对(+, *)

（1）因为E→E+E，且E ⇒ E*E，所以

$$+ \lessdot *$$

（2）因为E→E*E，且E ⇒ E+E，所以

$$+ \gtrdot *$$

G不是算符优先文法

示例

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): $P \rightarrow (E)$ $\therefore$ $(\doteq)$

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

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

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

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

由(4): $P \rightarrow (E)$, $E \Rightarrow E+T$ $\therefore$ $(\lessdot +$, $+ \gtrdot)$

. . .

$\therefore$ 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 $\doteq$ b的终结符对。

a $\doteq$ b 当且仅当G含有产生式 P →… ab… 或 P →… aQb…

- 确定满足关系 $\lessdot$ 和 $\gtrdot$ 的所有终结符对：

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

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

FIRSTVT(P) 和 LASTVT(P)

设 $P \in V_N$，定义：

FIRSTVT (P) =

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

LASTVT (P) =

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

FIRSTVT(P) 和 LASTVT(P)构造

■ FIRSTVT (P) 构造

- 规则1: 若 $P \rightarrow a \ldots$ 或 $P \rightarrow Qa \ldots$, 则 $a \in$FIRSTVT(P);
- 规则2: 若 $a \in$FIRSTVT(Q) , 且 $P \rightarrow Q \ldots$, 则 $a \in$FIRSTVT(P)。

■ LASTVT (P) 构造

- 规则1: 若 $P\rightarrow \ldots a$ 或 $P\rightarrow \ldots aQ$, 则 $a \in$ LASTVT(P);
- 规则2: 若 $a \in$ LASTVT(Q) , 且 $P\rightarrow \ldots Q$, 则 $a \in$ LASTVT(P)。

FIRSTVT(P) 的构造——数据结构

■ 二维布尔矩阵 F[P,a] 和符号栈STACK

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

栈 STACK：存放使FIRSTVT 为真的符号对 (P, a).

FIRSTVT(P)的构造——算法

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

示例

G: S→ a | ^ | (T)

T→T,S | S

求 FIRSTVT(S), FIRSTVT(T)

$$M=\begin{array}{c} \\ S \\ \\ T \end{array}\begin{array}{c} \begin{array}{ccccc} a & \wedge & (&) & , \end{array} \\ \left(\begin{array}{ccccc} T & T & T & F & F \\ & & & & \\ T & T & T & F & T \end{array}\right) \end{array}$$

T	,
T	(
T	^
T	a

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

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

类似地可得: LASTVT(S) ={a, ^,) }

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

FIRSTVT主程序:

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

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

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 $\lessdot$ 和 $\gtrdot$ 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 $a \lessdot b$。

  - **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 $a \gtrdot b$。

构造优先关系表算法

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

FOR i:=1 TO n-1 DO

BEGIN

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

IF $i \le n-2$且X_i和X_{i+2}都为终结符

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

IF X_i为终结符而X_{i+1}为非终结符 THEN

FOR FIRSTVT(X_{i+1})中的每个a DO

置 $X_i \lessdot a$;

IF X_i为非终结符而X_{i+1}为终结符 THEN

FOR LASTVT(X_i)中的每个a DO

置 $a \gtrdot X_{i+1}$

END

示例

G: $S \to a \mid \wedge \mid (T)$　　$T \to T,S \mid S$

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

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

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αβδ是文法G的一个句型，如果有

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

则称β是句型αβδ相对于非终结符A的短语。

示例

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。

示例

设文法G（S）： (1) S → aAcBe

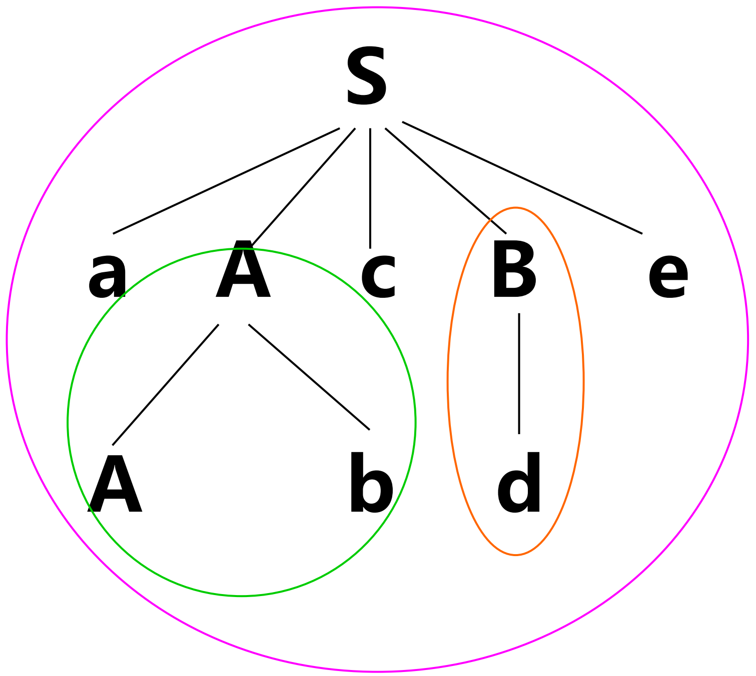
(2) A → b

(3) A → Ab

(4) B → d

给出句型aAbcde的短语。

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αβδ是文法G的一个句型，如果有

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

则称β是句型αβδ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。

示例

对文法G:

(1) E→E+T | T

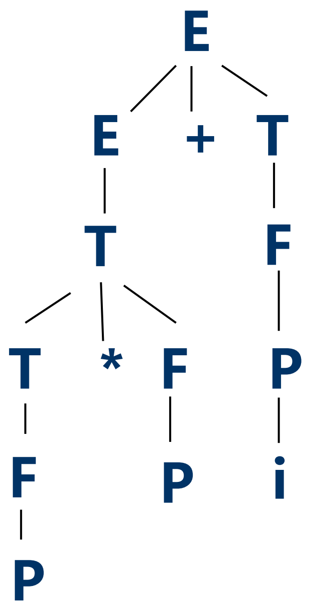
(2) T→T*F | F

(3) F→P↑F | P

(4) P→(E) | i

求句型P*P+i的短语、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

示例

对文法G:

(1) E→E+T | T

(2) T→T*F | F

(3) F→P↑F | P

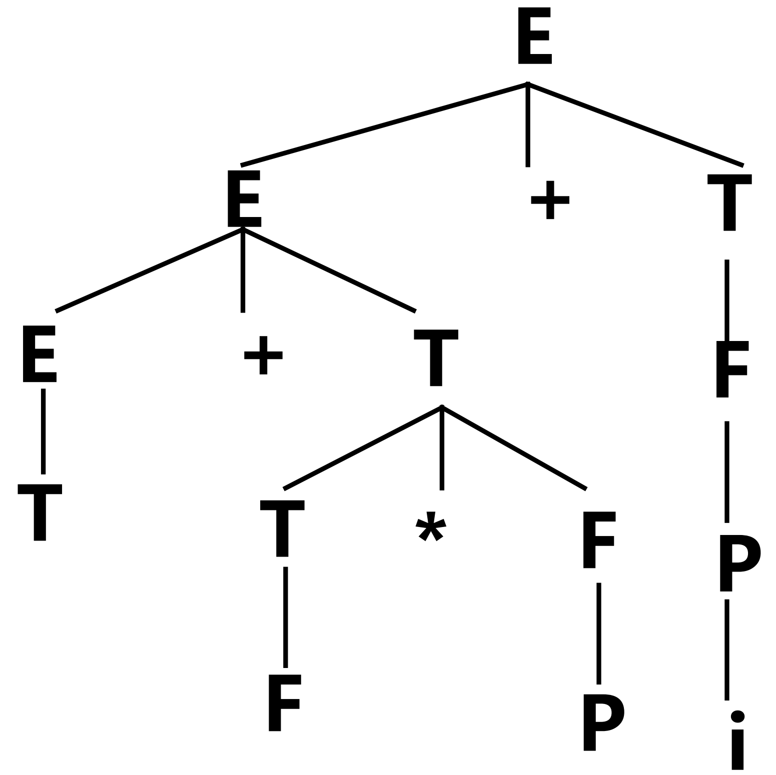
(4) P→(E) | i

句型： T+F*P+i

短语： T, F, P, i, F*P, T+F*P, T+F*P+i

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:

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

  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

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

  $a_{j-1} \lessdot a_j$

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

  $a_i \gtrdot a_{i+1}$

例. 句型#P*P+i#中，# ⋖ * , *⋗ +，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 $N_j a_j \dots\ N_i a_i\ N_{i+1}$

$$a_{j-1} \lessdot a_j$$

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

$$a_i \gtrdot a_{i+1}$$

$$\#\, N_1 a_1 \dots\ a_{j-1} \overbrace{N_j a_j \dots\ N_i a_i N_{i+1}}^{R} a_{i+1} \dots\ N_n a_n N_{n+1}\, \#$$

$$<\qquad = \qquad >$$

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

FIRSTVT

$$M=\begin{array}{cc} & \begin{array}{cccccc} + & * & \uparrow & (&) & i \end{array} \\ \begin{array}{c} E \\ T \\ F \\ P \end{array} & \begin{pmatrix} T & T & T & T & F & T \\ F & T & T & T & F & T \\ F & F & T & T & F & T \\ F & F & F & T & F & T \end{pmatrix} \end{array}$$

LASTVT

$$M=\begin{array}{cc} & \begin{array}{cccccc} + & * & \uparrow & (&) & i \end{array} \\ \begin{array}{c} E \\ T \\ F \\ P \end{array} & \begin{pmatrix} T & T & T & F & T & T \\ F & T & T & F & T & T \\ F & F & T & F & T & T \\ F & F & F & F & T & T \end{pmatrix} \end{array}$$

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

示例 对文法G: (1) E→E+T | T (2) T→T*F | F

(3) F→P↑F | P (4) P→(E) | i

句型：T+F*P+i

最左素短语： F*P

+ < * > +

	+	*	↑	(	)	i
+	>	<	<	<	>	<
*	>	>	<	<	>	<
↑	>	>	<	<	>	<
(	<	<	<	<	=	<
)	>	>	>		>	
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$S_{j+1} \ldots S_k$，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

```
k =1;   S[k]:=  " # " ;
REPEAT
   把下一个输入字符读进a中;
  IF  S[k]∈V_T  THEN  j:= k  ELSE  j:= k -1;
    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
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。

$$\begin{array}{ccccc} N \rightarrow & X_1 & X_2 & \dots & X_{k-j} \\ & \Updownarrow & \Updownarrow & & \Updownarrow \\ & S[j+1] & S[j+2] & \dots & S[k] \end{array}$$

x

示例

对文法G：(1) E→E+T | T　(2) T→T*F | F

(3) F→P↑F | P　(4) P→(E) | i

句子：i*(i+i)

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

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 #

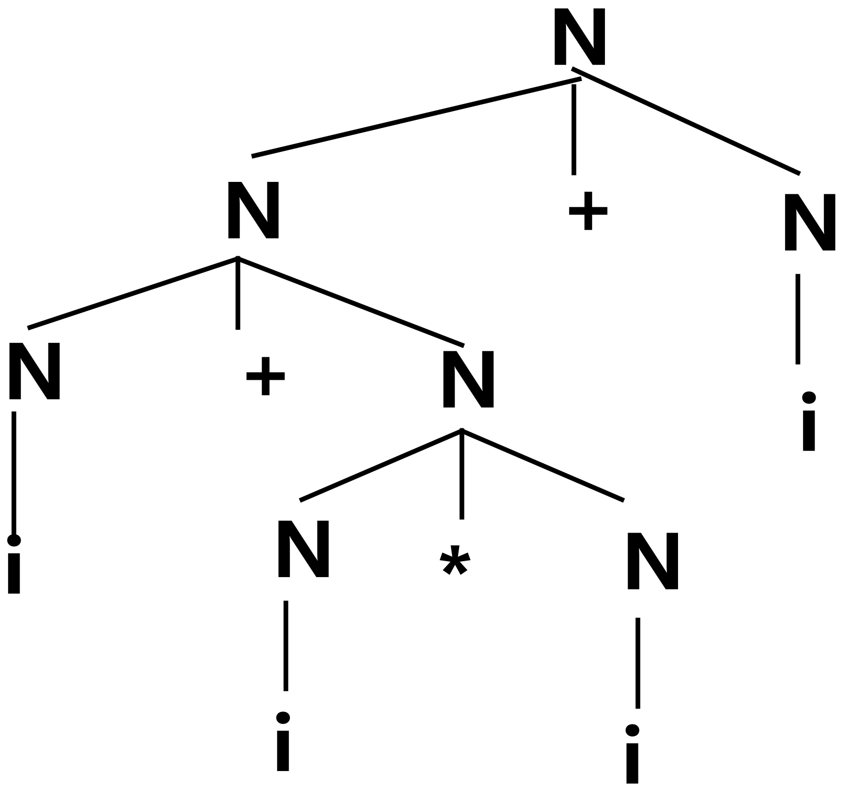
- 由于非终结符对归约没有影响，因此，非终结符可以不进符号栈S。

示例

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

3. 规范归约

4. LR分析方法

短语

- 令G是一个文法，S是文法的开始符号，若αβδ是文法G的一个句型，

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

  则称β是句型αβδ相对于非终结符A的**短语**。

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

- 一个句型的最左直接短语称为**句柄**。

示例

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。

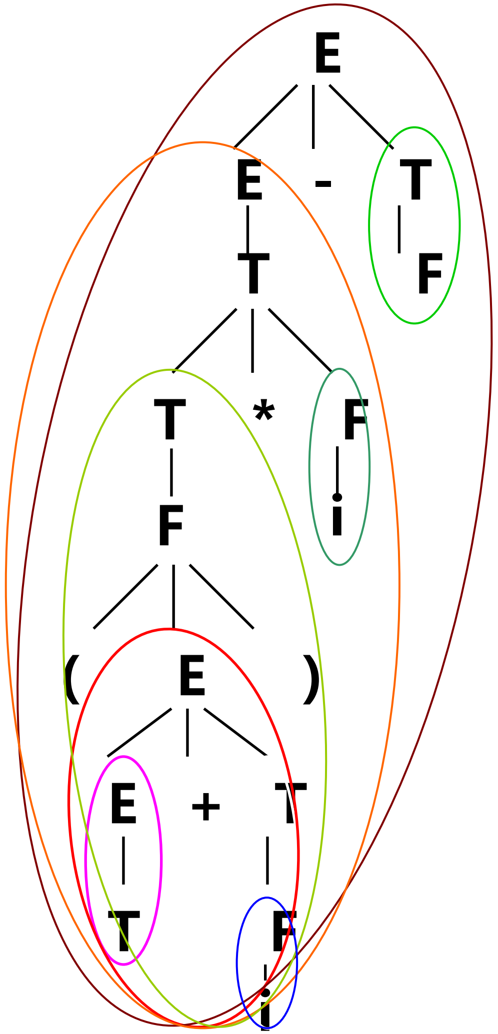
示例

G（E）： E→E+T | E-T | T

T→T*F | T/F | F

F→ i | (E)

试找出句型 (T+ i) * i-F 的所有短语、直接短语和句柄



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

示例 对文法G: (1) E→E+T | T (2) T→T*F | F

(3) F→P↑F | P (4) P→(E) | i

句型：T+F*P+i

短语： T, F, P, i , F*P,

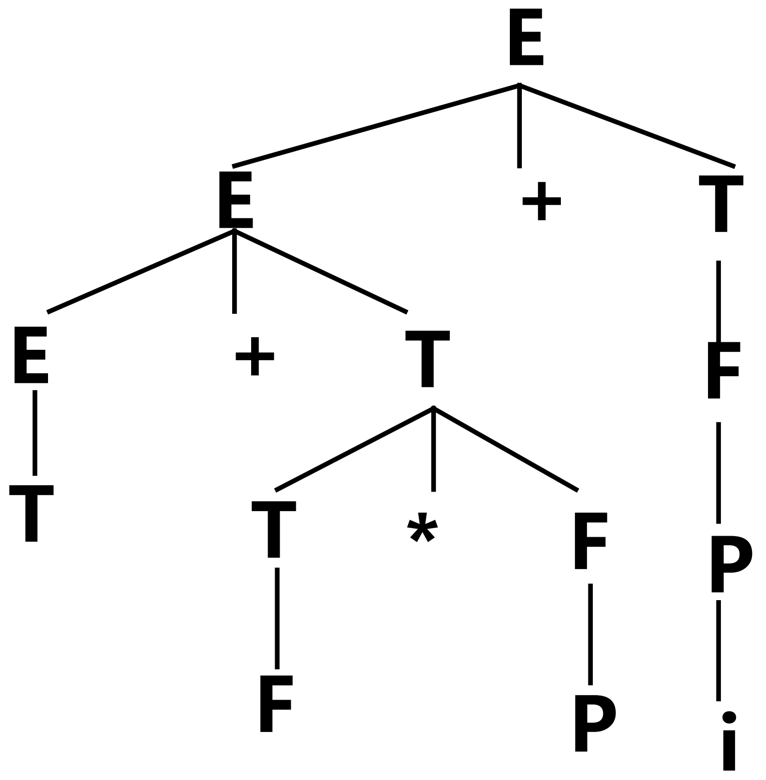
T+F*P , T+F*P+i

直接短语： T, F, P, i

句柄：T

素短语： F*P , i

最左素短语： F*P



示例

给定文法G：E→E+E|E*E|(E)|i

给出句型E+E*E的句柄

解.（1）E ⇒ E+E ⇒ E+E*E

E*E是句柄

（2）E ⇒ E*E ⇒ E+E*E

E+E是句柄

注: 二义性文法的句柄可能不唯一

规范归约

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

（1）$\alpha_n = \alpha$;

（2）$\alpha_0 = S$;

（3）对任意i，$0 < i \le n$，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 => aAcBe => aAcde =>aAbcde => a b b c d e

栈

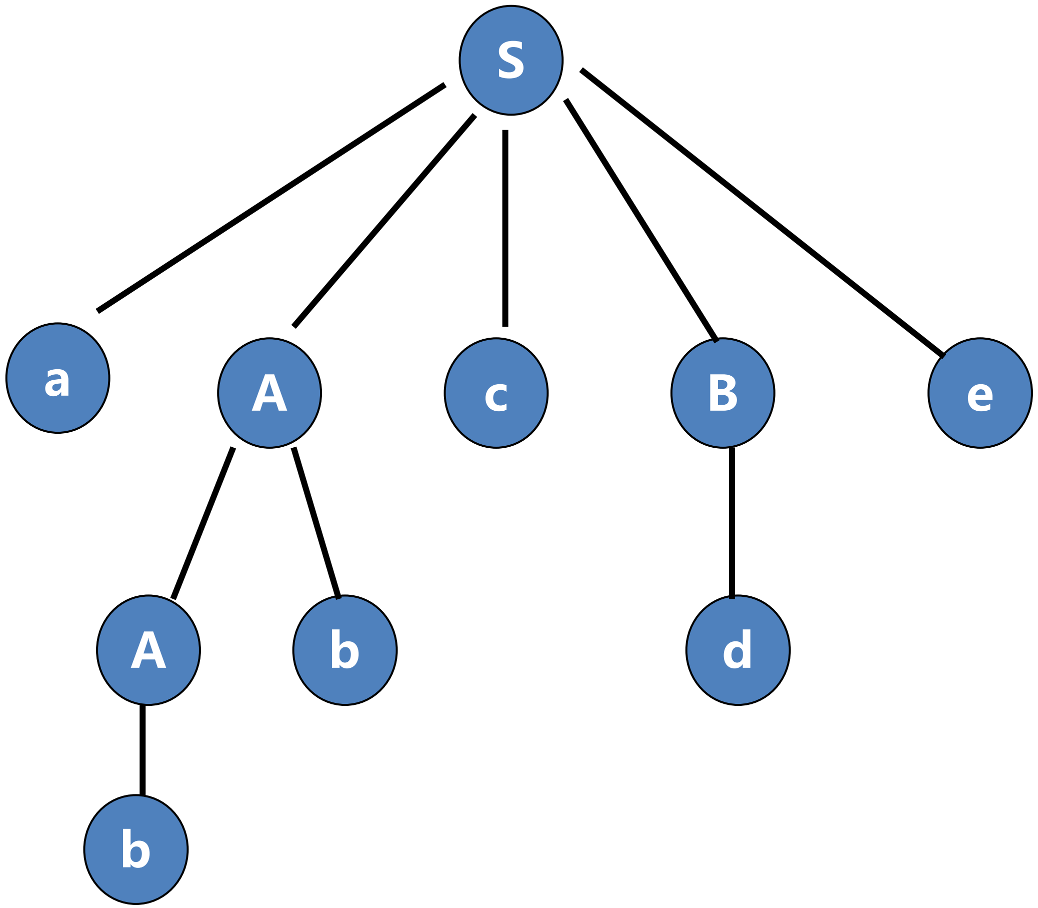
1	2	3	4	5	6	7	8	9	10
								e	
						d	B	B	
			b		c	c	c	c	
	b	A	A	A	A	A	A	A	
a	a	a	a	a	a	a	a	a	S

S->aAcBe A->Ab

A->b B->d

输入串：abbcde

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



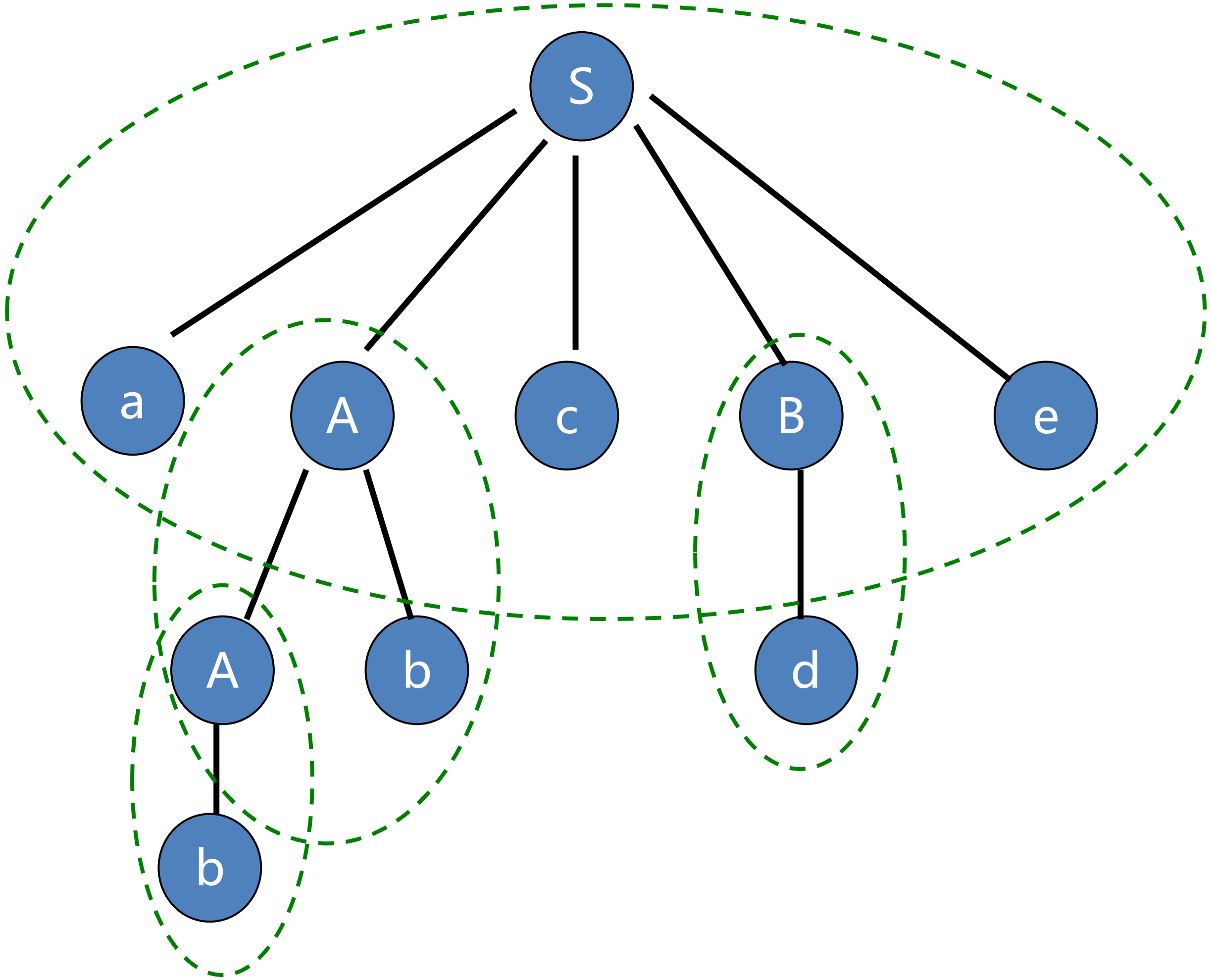
分 析 树

S->aAcBe   A->Ab

A->b   B->d

输入串：abbcde

句子abbcde 的规范归约过程如下: —— “ 剪枝”



文法G（S）：

S→aAcBe
A→b
A→Ab
B→ d

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

示例

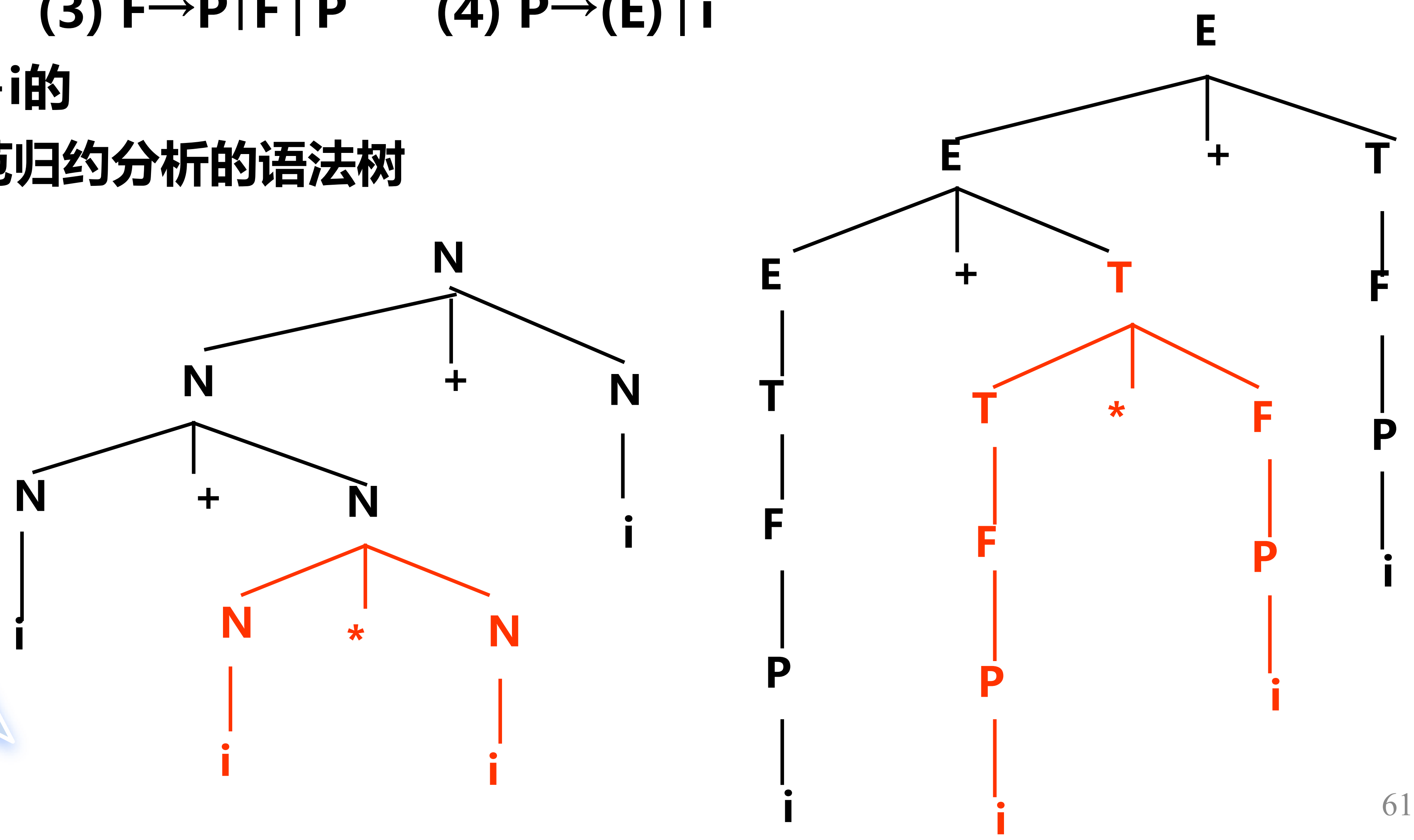
对文法G: (1) E→E+T | T (2) T→T*F | F

(3) F→P↑F | P (4) P→(E) | i

分别给出句子i+i*i+i的

算符优先分析和规范归约分析的语法树

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?