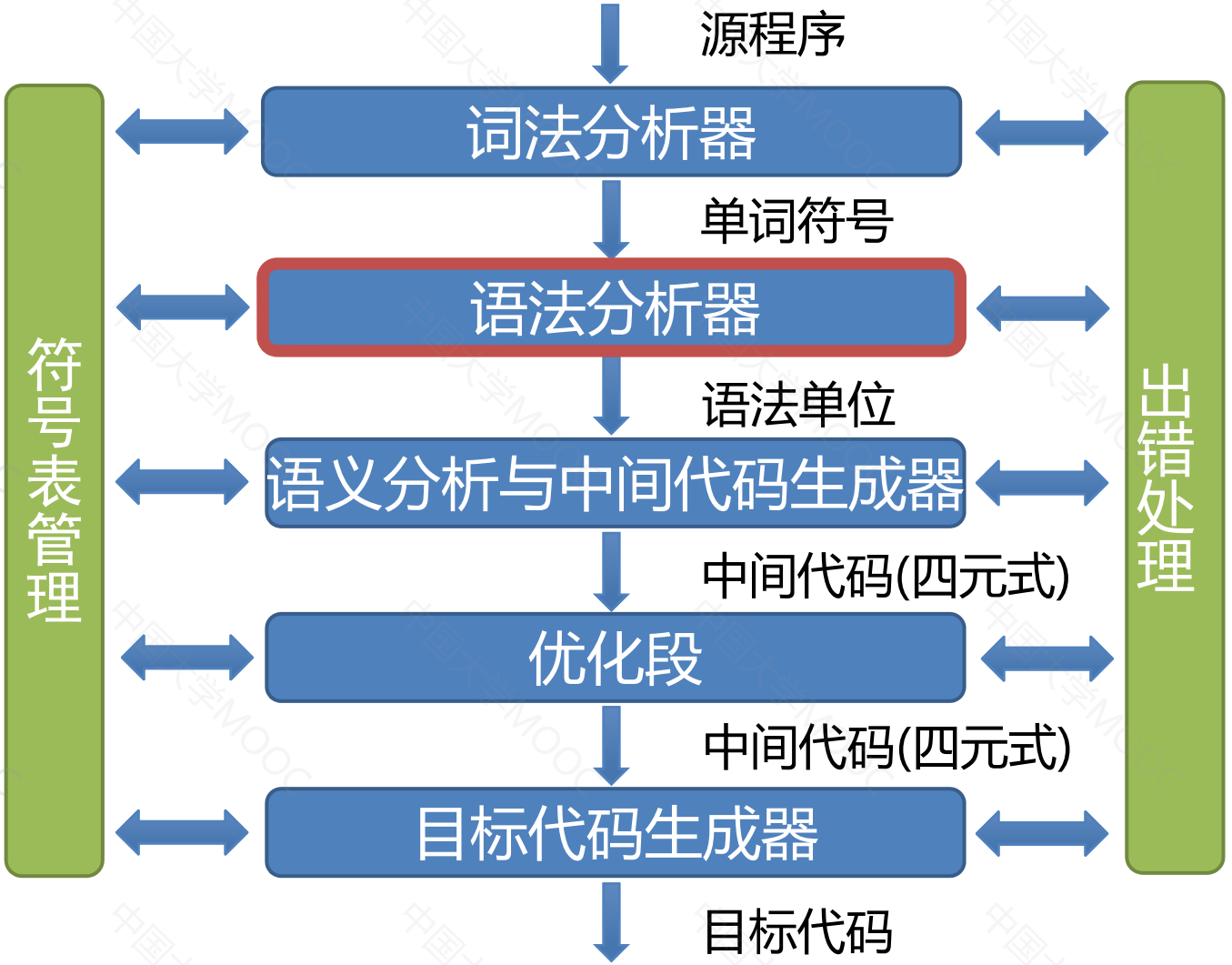
编译原理

自下而上分析的基本问题

编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
中间代码(四元式)
优化段
中间代码(四元式)
目标代码生成器
目标代码
符号表管理
出错处理

语法分析的方法

- 自上而下(Top-down)
  - 从文法的开始符号出发，反复使用各种产生式，寻找"匹配"的推导
  - 推导：根据文法的产生式规则，把串中出现的产生式的左部符号替换成右部
  - 从树的根开始，构造语法树
  - 递归下降分析法、预测分析程序
- 自下而上(Bottom-up)
  - 从输入串开始，逐步进行归约，直到文法的开始符号
  - 归约：根据文法的产生式规则，把串中出现的产生式的右部替换成左部符号
  - 从树叶节点开始，构造语法树
  - 算符优先分析法、LR分析法

自下而上分析示例

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

i*i+i

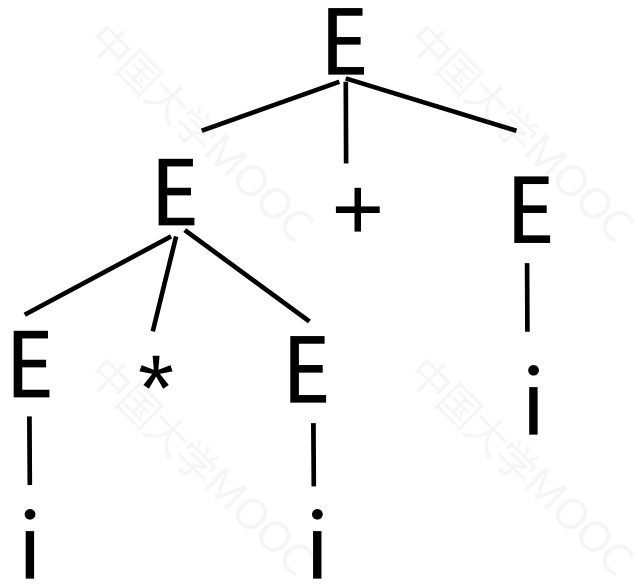
E*i+i

E*E+i

E+i

E+E

E



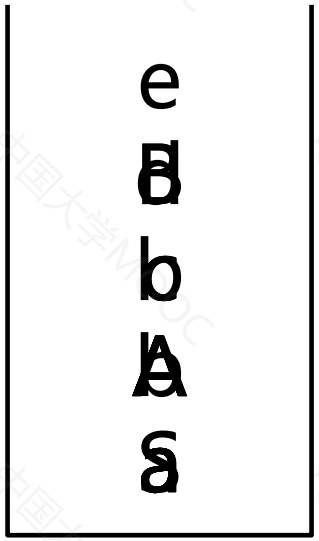
自下而上分析的基本思想

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

移进 - 归约分析示例

- 设文法G(S):
  (1) S → aAcBe
  (2) A → b
  (3) A → Ab
  (4) B → d

试对abbcde进行“移进 - 归约”分析。

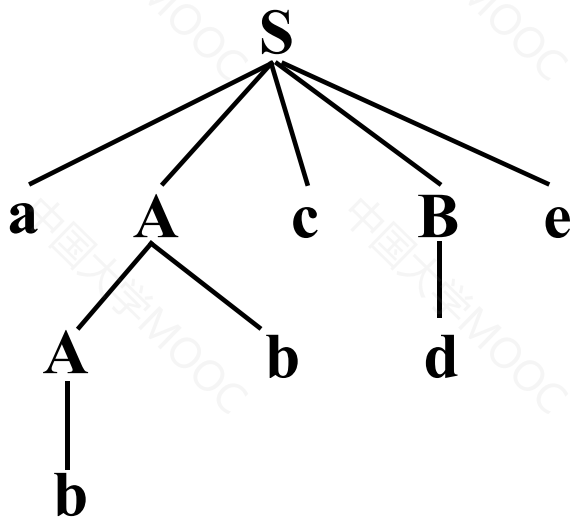


移进 - 归约分析示例

- 自下而上分析过程：边输入单词符号，边归约
- 核心问题：识别可归约串
- 分析树和语法树不一定一致

文法G(S)：
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d

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	a	a	S



测试：可归约串

- 你认为什么是可归约串？

A. 连续出现的单词序列

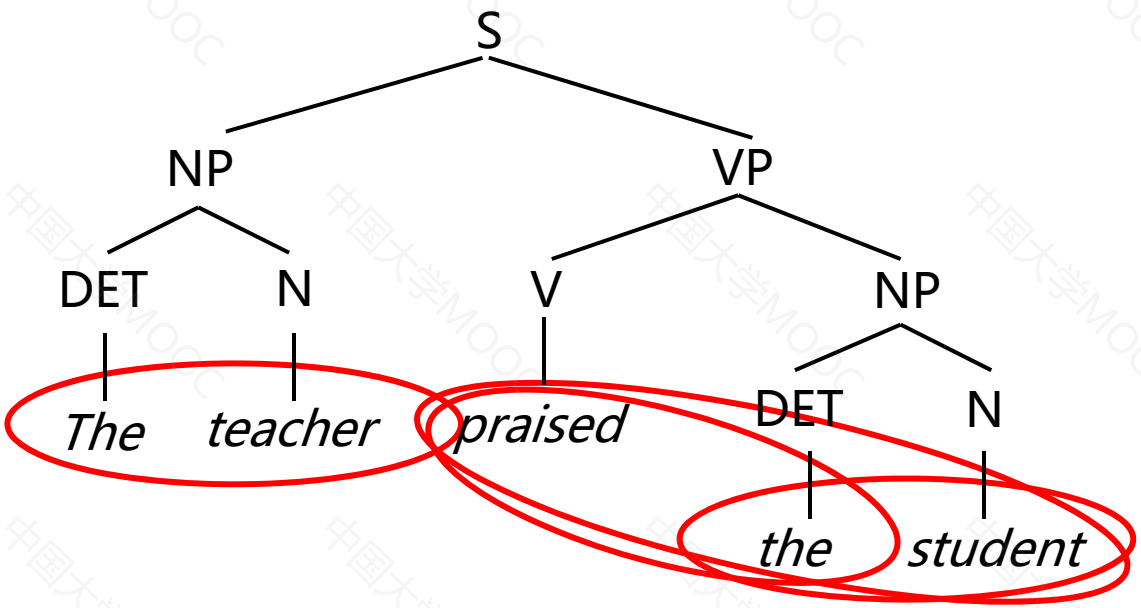
B. 短语

短语

- 定义：令G是一个文法，S是文法的开始符号，假定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β称是句型αβδ相对于非终结符A的短语。



短语

- 定义：令G是一个文法，S是文法的开始符号，假定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β称是句型αβδ相对于非终结符A的短语。

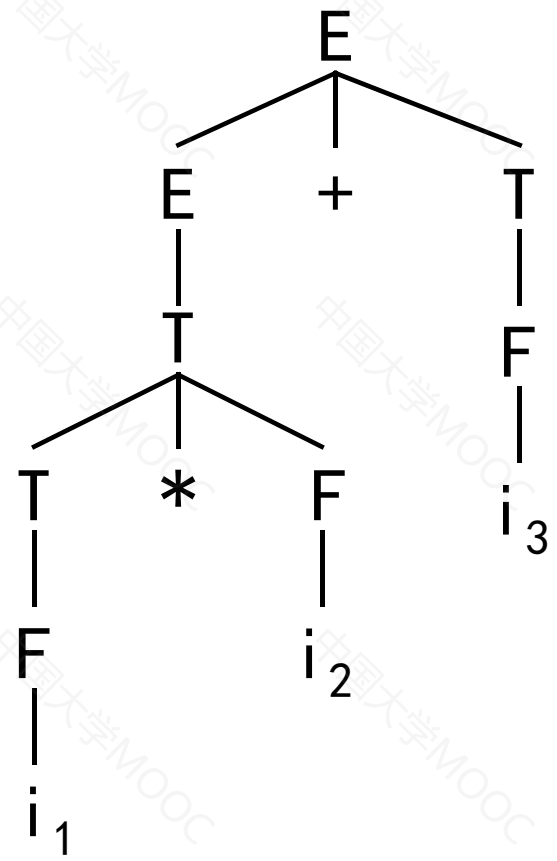
- 如果有$A \Rightarrow \beta$,则称β是句型αβδ相对于规则$A \rightarrow \beta$的直接短语。

测试：短语和直接短语

考虑文法G(E)：

$$E \rightarrow T \mid E+T$$
$$T \rightarrow F \mid T*F$$
$$F \rightarrow (E) \mid i$$

和句型$i_1*i_2+i_3$：



短语：i_1，i_2，i_3，i_1*i_2，$i_1*i_2+i_3$

直接短语：i_1，i_2，i_3

定义：令G是一个文法，S是文法的开始符号，假定αβδ是文法G的一个句型，如果有

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

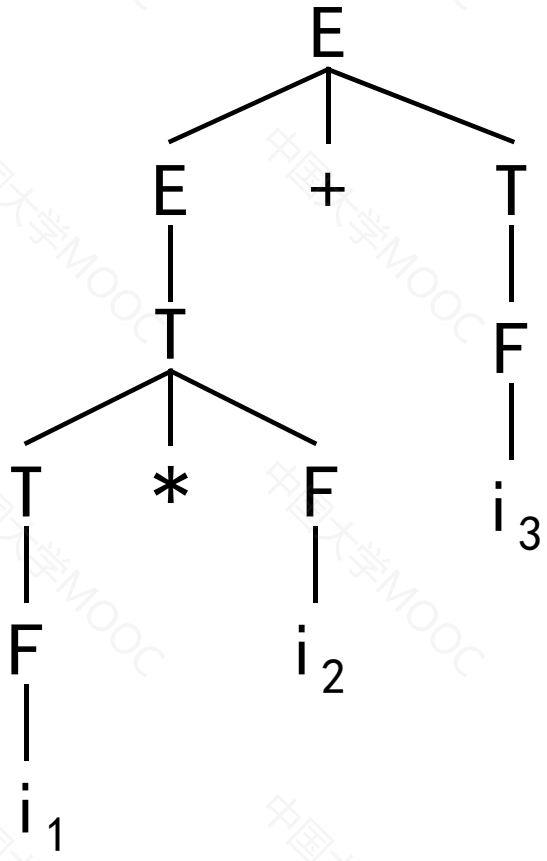
则β称是句型αβδ相对于非终结符A的短语。如果有$A \Rightarrow \beta$,则称β是句型αβδ相对于规则$A \rightarrow \beta$的直接短语。

短语和直接短语

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

短语：i_1，i_2，i_3，i_1*i_2，$i_1*i_2+i_3$

直接短语：i_1，i_2，i_3



分析过程描述

步骤	符号栈	输入串	所用产生式

- 考虑文法G(E)：

  $E \rightarrow T \mid E+T$

  $T \rightarrow F \mid T*F$

  $F \rightarrow (E) \mid i$

请给出$i_1*i_2+i_3$ 的分析过程。

小结

- 归约、移进-归约分析
- 核心问题：识别可归约串
- 短语、直接短语
- 分析过程的描述

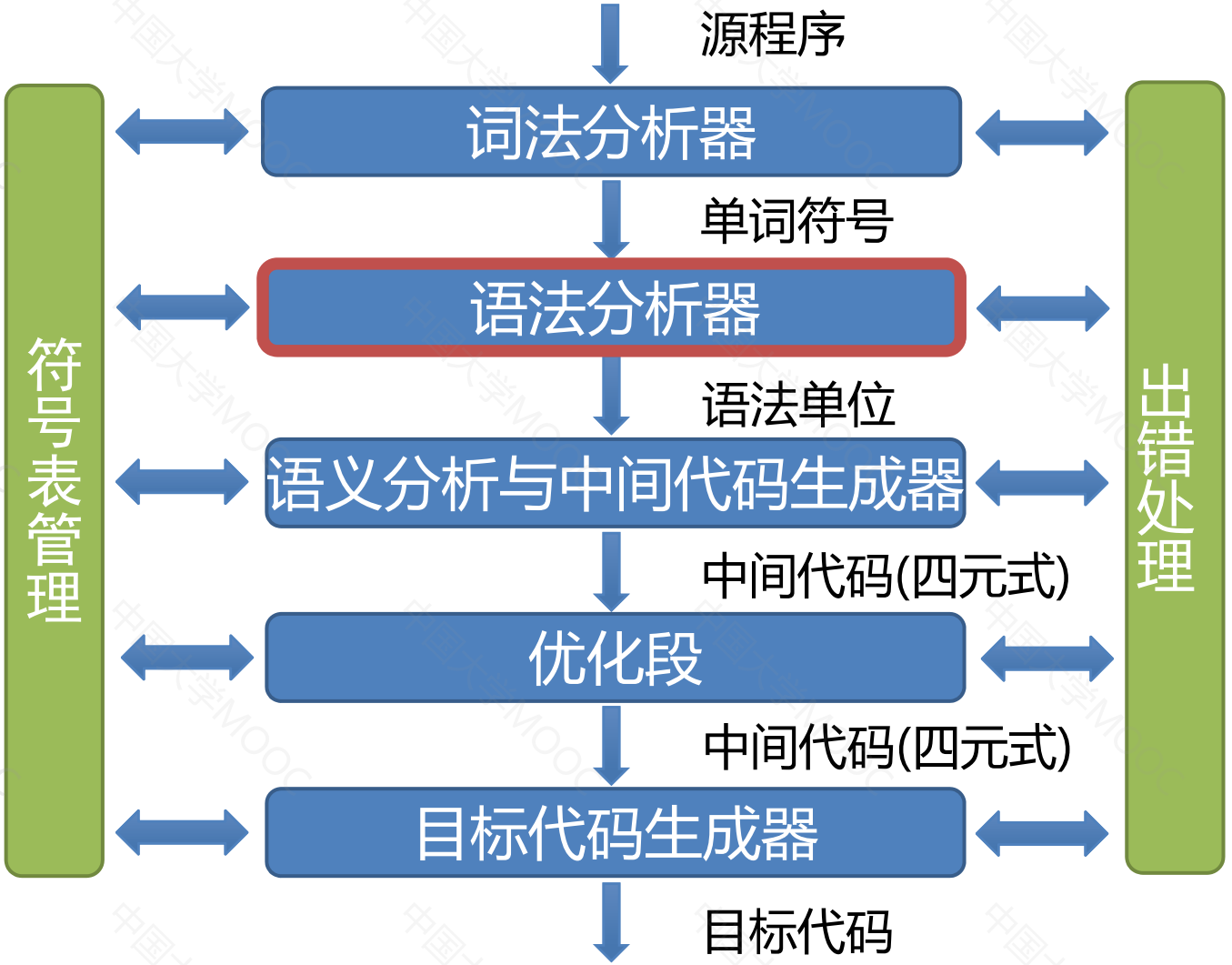
编译原理

算符优先分析方法

编译原理

自下而上分析法回顾

编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
中间代码(四元式)
优化段
中间代码(四元式)
目标代码生成器
目标代码
符号表管理
出错处理

自下而上分析法(Bottom-up)

- 基本思想
  - 从输入串开始，逐步归约，直到文法的开始符号
  - 归约：根据文法的产生式规则，把串中出现的产生式的右部替换成左部符号
  - 从树叶节点开始，构造语法树
- 算符优先分析法
  - 按照算符的优先关系和结合性质进行语法分析
  - 适合分析表达式
- LR分析法
  - 规范归约：句柄作为可归约串

编译原理

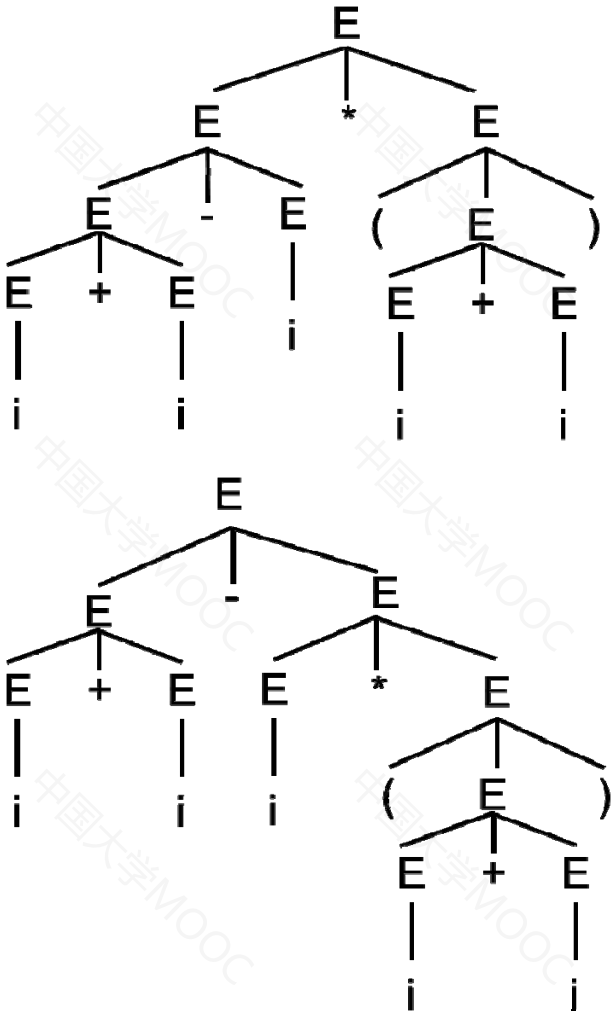
算符优先文法

运算的优先级

- 四则运算的优先规则
  - 先乘除后加减，同级从左到右
- 考虑文法G'(E)：

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

- 句子i+i-i*(i+i) 有几种不同的归约。
- 归约顺序不同，计算的顺序也不同，结果也不一样
- 如果规定算符的优先次序，并按这种规定进行归约，则归约过程是唯一的



句子i+i-i*(i+i)的归约过程

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

(1) i+i-i*(i+i)
(2) F+i-i*(i+i)
(3) T+i-i*(i+i)
(4) E+i-i*(i+i)
(5) E+F-i*(i+i)
(6) E+T-i*(i+i)
(7) E-i*(i+i)
(8) E-F*(i+i)
(9) E-T*(i+i)
(10) E-T*(F+i)
(11) E-T*(T+i)
(12) E-T*(E+i)
(13) E-T*(E+F)
(14) E-T*(E+T)
(15) E-T*(E)
(16) E-T*F
(17) E-T
(18) E

G'(E):

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

G(E): $E \rightarrow T \mid E+T \mid E-T$

$T \rightarrow F \mid T*F \mid T/F$

$F \rightarrow (E) \mid i$

优先关系

- 任何两个可能相继出现的终结符a与b可能三种优先关系
  - 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
- 算符优先关系与数学上的<>=不同
  - \+ $\lessdot$ +
  - a $\lessdot$ b并不意味着b $\gtrdot$ a，如($\lessdot$ +　和　+ $\lessdot$ (

算符文法

- 一个文法，如果它的任一产生式的右部都不含两个相继(并列)的非终结符，即不含...QR...形式的产生式右部，则我们称该文法为算符文法。
- 约定：
  - a、b代表任意终结符
  - P、Q、R代表任意非终结符
  - ‘...’代表由终结符和非终结符组成的任意序列，包括空字

G’(E)：

E → i | E+E | E-E | E*E | E/E | (E)

G(E)：E → T | E+T | E-T

T → F | T*F | T/F

F → (E) | i

算符优先文法

G'(E)：

E → i | E+E | E-E | E*E | E/E | (E)

G(E)：E → T | E+T | E-T

T → F | T*F | T/F

F → (E) | i

- 假定G是一个不含ε-产生式的算符文法。对于任何一对终结符a、b，我们说：
  1. $a \doteq b$，当且仅当文法G中含有形如P→...ab...或P→...aQb...的产生式；
  2. $a \lessdot b$，当且仅当G中含有形如P→...aR...的产生式，而$R \overset{+}{\Rightarrow} b...$或$R \overset{+}{\Rightarrow} Qb...$；
  3. $a \gtrdot b$，当且仅当G中含有形如P→...Rb...的产生式，而 $R \overset{+}{\Rightarrow} ...a$或$R \overset{+}{\Rightarrow} ...aQ$。
- 如果一个算符文法G中的任何终结符对(a，b)至多只满足$a \doteq b$、$a \lessdot b$和 $a \gtrdot b$这三个关系之一，则称G是一个算符优先文法。

示例：算符优先文法

- 考虑下面的文法G(E)：
  (1) E→E+T | T
  (2) T→T*F | F
  (3) F→P↑F | P
  (4) P→(E) | i
- 由规则P→(E)，有($\doteq$)
- 由规则E→E + T和T⇒T*F，有 + $\lessdot$ *
- 由T→T*F 和F ⇒ P ↑ F，可得* $\lessdot$ ↑
- 由E→E + T和E ⇒ E+T，可得 + $\gtrdot$ +
- 由F→P↑F和F ⇒ P↑F，可得 ↑ $\lessdot$ ↑
- 由P→(E)和
  E⇒E+T⇒T+T⇒T*F+T⇒F*F+T⇒P↑F*F+T⇒i↑F*F+T
  有 ($\lessdot$ +　($\lessdot$ *　($\lessdot$ ↑　($\lessdot$ i

示例：算符优先文法

- 文法G(E)：

(1) E→E+T | T
(2) T→T*F | F
(3) F→P ↑ F | P
(4) P→(E) | i
的优先关系表：

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

编译原理

构造优先关系表的算法
——FIRSTVT和LASTVT集合

构造优先关系表的算法

- 确定满足关系$\doteq$的所有终结符对
  - a$\doteq$b，当且仅当文法G中含有形如P→...ab...或P→...aQb...的产生式
  - 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

$FIRSTVT(P) = \{a \mid P \overset{+}{\Rightarrow} a \ldots$ 或 $P \overset{+}{\Rightarrow} Q\,a \ldots,\ a \in V_T$且$Q \in V_N\}$

  - a$\lessdot$b，当且仅当G中含有形如P→...aR...的产生式，而R$\overset{+}{\Rightarrow}$b...或R$\overset{+}{\Rightarrow}$Qb...；
  - a$\gtrdot$b，当且仅当G中含有形如P→...Rb...的产生式，而 R$\overset{+}{\Rightarrow}$...a或R$\overset{+}{\Rightarrow}$...aQ。

$LASTVT(P) = \{a \mid P \overset{+}{\Rightarrow} \ldots a$ 或 $P \overset{+}{\Rightarrow} \ldots a\,Q,\ a \in V_T$且$Q \in V_N\}$

构造优先关系表的算法

- 根据FIRSTVT和LASTVT集合，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

构造集合FIRSTVT(P)的算法

- 反复使用下面两条规则构造集合FIRSTVT(P)

1. 若有产生式P→a…或P→Qa…，则
   a∈FIRSTVT(P)

2. 若a∈FIRSTVT(Q)，且有产生式P→Q…，则
   a∈FIRSTVT(P)

$FIRSTVT(P)=\{a \mid P \overset{+}{\Rightarrow} a\ldots$ 或 $P \overset{+}{\Rightarrow} Q\,a\ldots,\ \ a \in V_T$且$Q \in V_N\}$

构造集合FIRSTVT(P)的算法

- 算法的一种实现
  - 布尔数组F[P，a]，使得F[P，a]为真的条件是，当且仅当a∈ FIRSTVT(P)。开始时，按上述的规则1对每个数组元素F[P，a]赋初值。
  - 栈STACK，把所有初值为真的数组元素F[P，a]的符号对(P，a)全都放在STACK之中。

1. 若有产生式P→a…或P→Qa…，则a∈ FIRSTVT(P)

2. 若a∈ FIRSTVT(Q)，且有产生式P→Q…，则a∈ FIRSTVT(P)

构造集合FIRSTVT(P)的算法

- 算法的一种实现
  - 若栈STACK不空，就将栈顶项弹出，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。
  - 上述过程一直重复，直至栈STACK为空为止。

1. 若有产生式P→a…或P→Qa…，则a∈FIRSTVT(P)

2. 若a∈FIRSTVT(Q)，且有产生式P→Q…，则a∈FIRSTVT(P)

构造集合FIRS

1. 若有产生式P→a...或P→Qa...，则a∈FIRSTVT(P)
2. 若a∈FIRSTVT(Q)，且有产生式P→Q...，则a∈FIRSTVT(P)

- 伪码实现

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

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

构造集合FIRSTVT(P)的算法

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

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

构造集合LASTVT(P)的算法

- 反复使用下面两条规则构造集合LASTVT(P)

1. 若有产生式P→… a或P→ … aQ，则 a∈LASTVT(P)

2. 若a∈LASTVT(Q)，且有产生式P→… Q，则 a∈LASTVT(P)

$$LASTVT(P) = \{ a \mid P \overset{+}{\Rightarrow} \ldots a \text{ 或 } P \overset{+}{\Rightarrow} \ldots a\,Q\,,\ \ a \in V_T \text{且} Q \in V_N \}$$

编译原理

构造优先关系表的算法
——FIRSTVT和LASTVT集合计算示例

示例： FIRSTVT和LASTVT的计算

- 考虑下面的文法G(E)：
  (1) E→E+T | T
  (2) T→T*F | F
  (3) F→P ↑ F | P
  (4) P→(E) | i

计算文法G的FIRSTVT和LASTVT。

示例： FIRSTVT和LASTVT的计算

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

FIRSTVT

	+	*	↑	(	)	i
E	√	√	√	√		√
T		√	√	√		√
F			√	√		√
P				√		√

1. 若有产生式P→a…或P→Qa…，则a∈FIRSTVT(P)
2. 若a∈FIRSTVT(Q)，且有产生式P→Q…，则a∈FIRSTVT(P)

FIRSTVT(E)={ +, *, ↑, (, i }
FIRSTVT(T)={ *, ↑, (, i }
FIRSTVT(F)={ ↑, (, i }
FIRSTVT(P)={ (, i }

示例： FIRSTVT和LASTVT的计算

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

LASTVT

	+	*	↑	(	)	i
E	√	√	√		√	√
T		√	√		√	√
F			√		√	√
P					√	√

1. 若有产生式P→… a或P→ … aQ，则 a∈LASTVT(P)
2. 若a∈LASTVT(Q)，且有产生式P→… Q，则a∈LASTVT(P)

LASTVT(E)={ +, *, ↑,), i }
LASTVT(T)={ *, ↑,), i }
LASTVT(F)={ ↑,), i }
LASTVT(P)={), i }

编译原理

构造优先关系表的算法

构造优先关系表的算法

- 通过检查G的每个产生式的每个候选式，可找出所有满足a$\doteq$b的终结符对。
- 根据FIRSTVT和LASTVT集合，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

构造优先关系表的算法

FOR 每条产生式$P \to X_1X_2 \ldots X_n$ 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

$P \rightarrow X_1 X_2 \cdots X_i X_{i+1} X_{i+2} \cdots X_n$

构造优先关系表的算法

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≤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

$P \rightarrow X_1 X_2 \cdots X_i X_{i+1} X_{i+2} \cdots X_n$

构造优先关系表的算法

FOR 每条产生式$P \rightarrow X_1X_2 \ldots X_n$ DO
  FOR i:=1 TO n-1 DO
  BEGIN
    IF X_i和X_{i+1}均为终结符 THEN 置$X_i \doteq X_{i+1}$
    IF $i \leq n-2$且X_i和X_{i+2}都为终结符，但X_{i+1}为非终结符 THEN 置$X_i \doteq X_{i+2}$；
    IF X_i为终结符而X_{i+1}为非终结符 THEN
      FOR FIRSTVT(X_{i+1})中的每个a DO
        置 $X_i \lessdot a$；
    IF X_i为非终结符而X_{i+1}为终结符 THEN
      FOR LASTVT(X_i)中的每个a DO
        置 $a \gtrdot X_{i+1}$
  END

$P \rightarrow X_1 X_2 \cdots X_i X_{i+1} X_{i+2} \cdots X_n$

构造优先关系表的算法

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≤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

$P \rightarrow X_1 X_2 \cdots X_i X_{i+1} X_{i+2} \cdots X_n$

构造优先关系表的算法

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≤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

编译原理

构造优先关系表示例

示例：构造优先关系表

- 考虑下面的文法G(E)：
  (1) E→E+T | T
  (2) T→T*F | F
  (3) F→P ↑ F | P
  (4) P→(E) | i

1. 计算文法G的FIRSTVT和LASTVT；
2. 构造优先关系表;
3. G是算符优先文法吗?

示例：构造优先关系表

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

- 计算文法G的FIRSTVT和LASTVT
- 构造构造优先关系表
- G是算符优先文法

FIRSTVT(E)={ +, *, ↑, (, i }
FIRSTVT(T)={ *, ↑, (, i }
FIRSTVT(F)={ ↑, (, i }
FIRSTVT(P)={ (, i }

LASTVT(E)={ +, *, ↑,), i }
LASTVT(T)={ *, ↑,), i }
LASTVT(F)={ ↑,), i }
LASTVT(P)={), i }

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

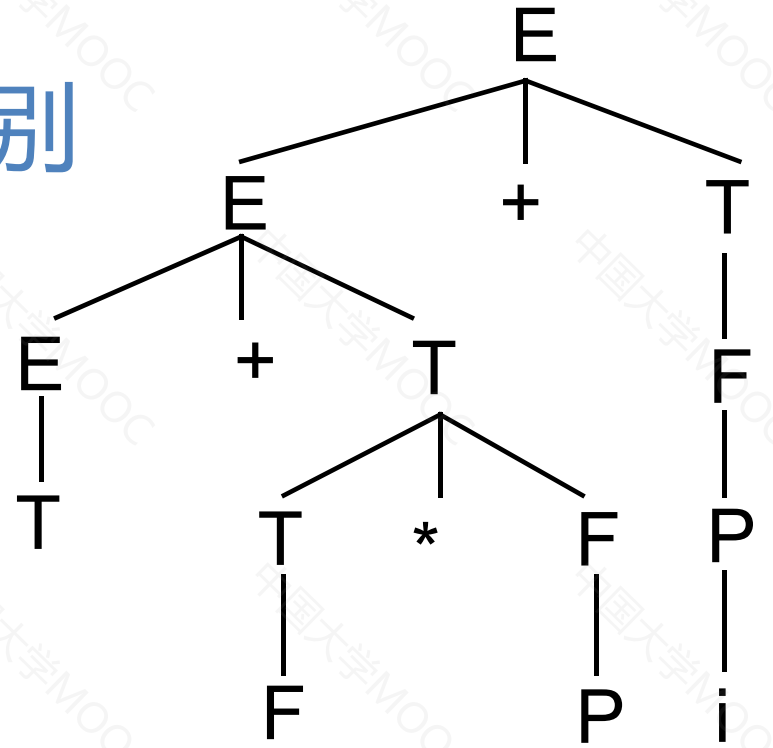
编译原理

算符优先分析算法

最左素短语

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

示例：各类短语的识别



- 考虑文法G(E)：

(1) E→E+T | T
(2) T→T*F | F
(3) F→P ↑ F | P
(4) P→(E) | i

对于句型：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

素短语： i，F*P

最左素短语： F*P

最左素短语

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

最左素短语定理

- 算符优先文法句型(括在两个＃之间)的一般形式：

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

其中，a_i都是终结符，N_i是可有可无的非终结符。

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

$a_{j-1} \lessdot a_j$

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

$a_i \gtrdot a_{i+1}$

$$\#N_1a_1N_2a_2\ldots\ a_{j-1}\ N_j\ a_j\ldots N_i\ a_i\ N_{i+1}\ a_{i+1}\ \ldots N_na_nN_{n+1}\#$$

算符优先分析算法

- 使用一个符号栈S，用它寄存终结符和非终结符，k代表符号栈S的使用深度
- 在正确的情况下，算法工作完毕时，符号栈S应呈现：# N #

$\#N_1a_1N_2a_2\ldots\ a_{j-1}\ N_j\ a_j\ldots N_i\ a_i\ N_{i+1}\ a_{i+1}\ \ldots N_na_nN_{n+1}\#$

```
k:=1;
S[k]:='#';
REPEAT
    把下一个输入符号读进a中;
    IF S[k]∈VT THEN  j:=k  ELSE  j:=k-1;
    WHILE S[j]⋗a DO
    BEGIN
        REPEAT
            Q:=S[j];
            IF S[j-1]∈VT THEN j:=j-1
                         ELSE j:=j-2
        UNTIL S[j]⋖Q;
        把S[j+1]…S[k]归约为某个N;
        k:=j+1;
        S[k]:=N
    END OF WHILE;
    IF S[j]⋖a OR S[j]≐a THEN
            BEGIN k:=k+1;S[k]:=a END
    ELSE ERROR /*调用出错诊察程序*/
UNTIL a='#'
```

$\#N_1a_1N_2a_2\ldots\ a_{j-1}\ N_j\ a_j\ldots N_i\ a_i\ N_{i+1}\ a_{i+1}\ \ldots N_na_nN_{n+1}\#$

```
k:=1;
S[k]:='#';
REPEAT
    把
    IF
    W
    B
                                    ELSE j:=j-2
        UNTIL S[j]⋖Q;
        把S[j+1]…S[k]归约为某个N；
        k:=j+1;
        S[k]:=N
    END OF WHILE;
    IF S[j]⋖a OR S[j]≐a THEN
            BEGIN k:=k+1;S[k]:=a END
    ELSE ERROR /*调用出错诊察程序*/
UNTIL a='#'
```

自左至右，终结符对终结符，非终结符对非终结符，而且对应的终结符相同。

$$N \rightarrow \underset{\updownarrow}{X_1}\ \underset{\updownarrow}{X_2}\ \ldots\ \underset{\updownarrow}{X_{k-j}}$$

$$S[j+1]\ S[j+2]\ \ldots\ S[k]$$

分析栈

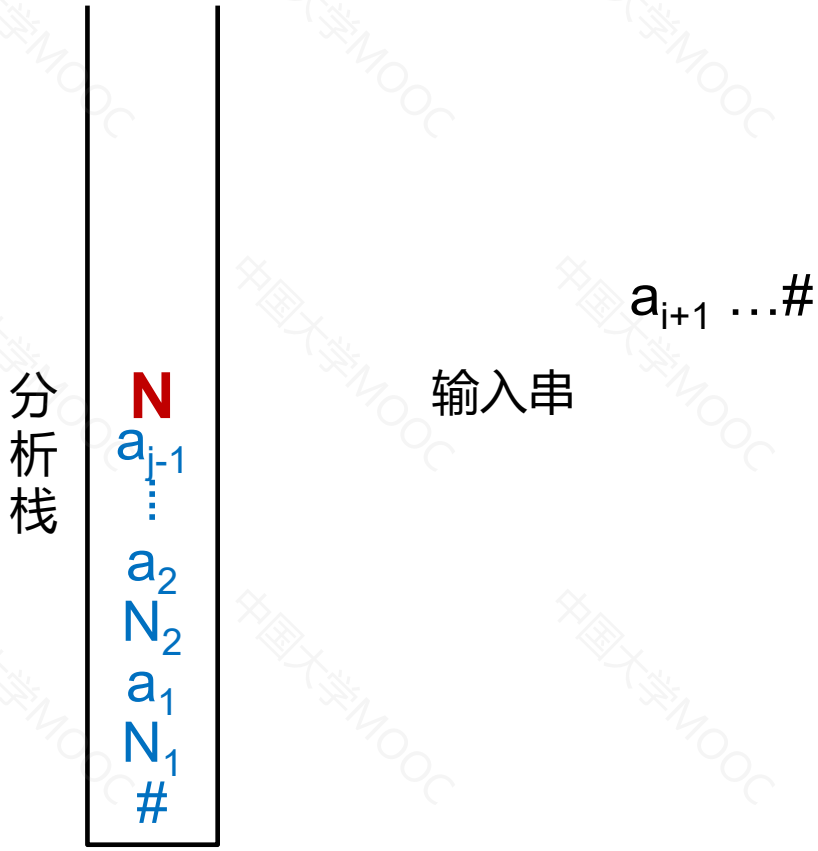
N_{i+1}
a_i
N_i
⋮
a_j
N_j
a_{j-1}
⋮
a_2
N_2
a_1
N_1
#

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

输入串

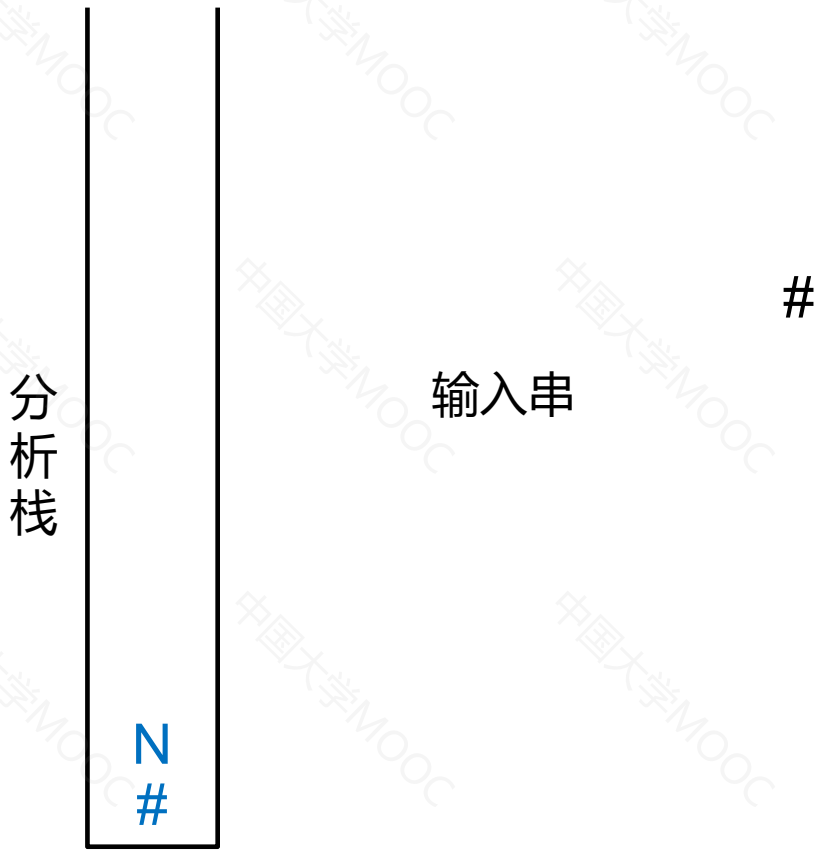
$\#N_1a_1N_2a_2\cdots a_{j-1}\ N_j\ a_j\ldots N_i\ a_i\ N_{i+1}\ a_{i+1}\ \ldots N_na_nN_{n+1}\#$

```
k:=1;
S[k]:='#';
REPEAT
    把下一个输入符号读进a中;
    IF S[k]∈VT THEN  j:=k  ELSE  j:=k-1;
    WHILE S[j]⋗a DO
    BEGIN
        REPEAT
            Q:=S[j];
            IF S[j-1]∈VT THEN j:=j-1
                         ELSE j:=j-2
        UNTIL S[j]⋖Q;
        把S[j+1]…S[k]归约为某个N;
        k:=j+1;
        S[k]:=N
    END OF WHILE;
    IF S[j]⋖a OR S[j]≐a THEN
            BEGIN k:=k+1;S[k]:=a END
    ELSE ERROR /*调用出错诊察程序*/
UNTIL a='#'
```



$\#N_1a_1N_2a_2\cdots a_{j-1}\ N_j\ a_j\ldots N_i\ a_i\ N_{i+1}\ a_{i+1}\ \ldots N_na_nN_{n+1}\#$

```
k:=1;
S[k]:='#';
REPEAT
    把下一个输入符号读进a中;
    IF S[k]∈VT THEN  j:=k  ELSE  j:=k-1;
    WHILE S[j]⋗a DO
    BEGIN
        REPEAT
            Q:=S[j];
            IF S[j-1]∈VT THEN j:=j-1
                         ELSE j:=j-2
        UNTIL S[j]⋖Q;
        把S[j+1]…S[k]归约为某个N;
        k:=j+1;
        S[k]:=N
    END OF WHILE;
    IF S[j]⋖a OR S[j]≐a THEN
            BEGIN k:=k+1;S[k]:=a END
    ELSE ERROR /*调用出错诊察程序*/
UNTIL a='#'
```



测试：分析树 vs.语法树

- 对于文法的句子来说，它的算符优先分析的结果就是语法树

  A. 正确

  B. 错误

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。

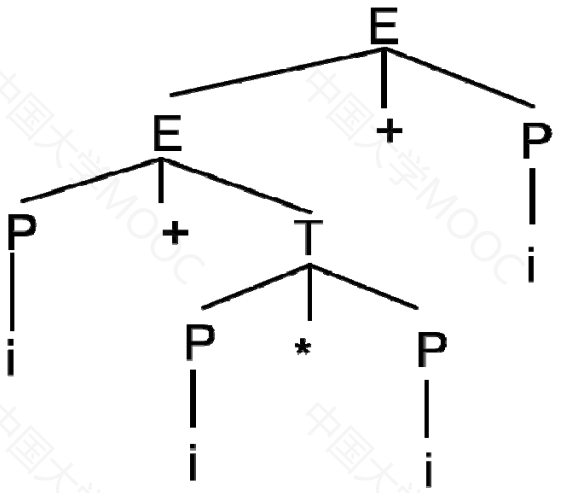
分析树 vs.语法树

- 算符优先分析结果不一定是语法树

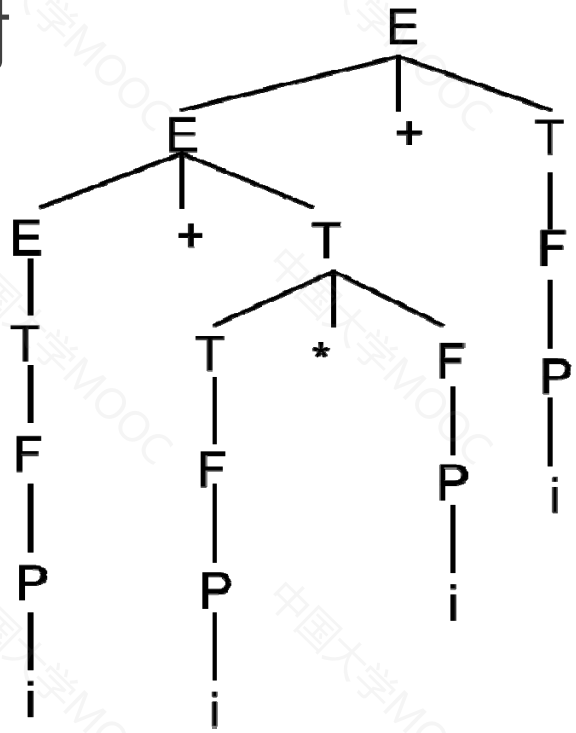
考虑文法G(E):

(1) E→E+T | T
(2) T→T*F | F
(3) F→P ↑ F | P
(4) P→(E) | i

的句子i+i*i+i



算符优先分析得到的分析树

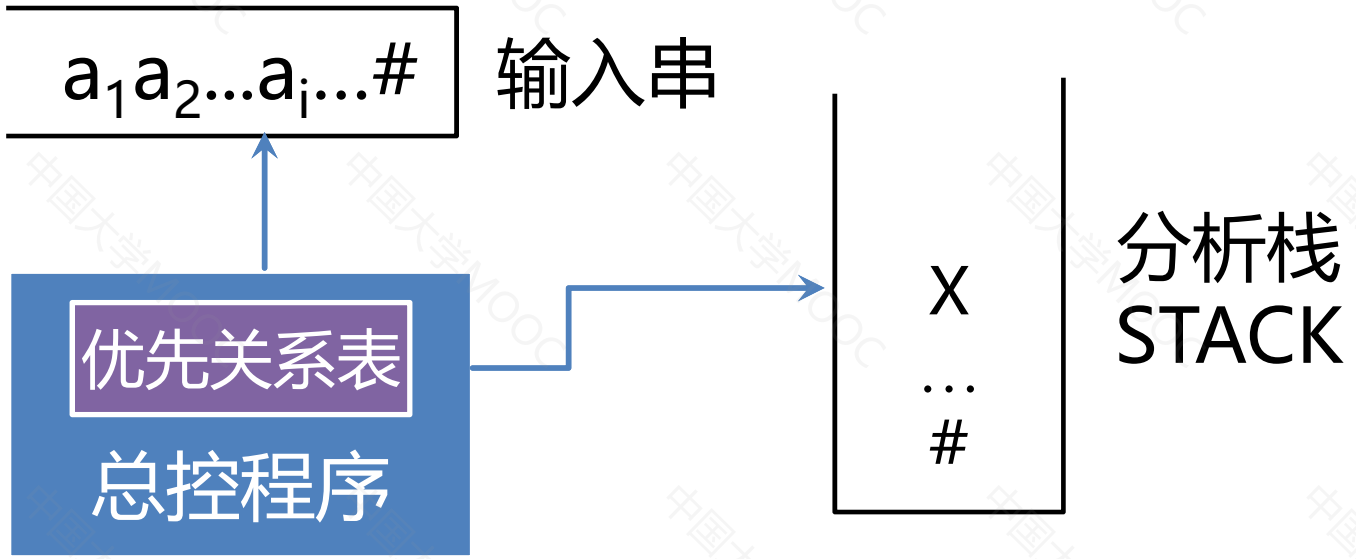


语法树

- 计算思维的典型方法
  - 知识与控制的分离
  - 自动化

算符优先分析程序构成

- 总控程序，根据现行栈顶符号和当前输入符号，执行动作
- 优先关系表，用于指导总控程序进行移进-归约
- 分析栈 STACK，用于存放文法符号



算符优先分析法

- 特点
  - 优点：简单，快速
  - 缺点：可能错误接受非法句子
- 使用广泛
  - 用于分析各类表达式
  - ALGOL 60

小结

- 算符文法与算符优先文法
- 优先关系
  - 计算FIRSTVT和LASTVT集合
  - 构造算符优先关系表
- 算符优先分析算法
  - 最左素短语