



*School of Computer Science & Technology*  
*Harbin Institute of Technology*



# 第四章 自顶向下的 语法分析



**重点：**自顶向下分析的基本思想，预测分析器总体结构，预测分析表的构造，递归下降分析法基本思想，简单算术表达式的递归下降分析器。

**难点：**FIRST 和 FOLLOW 集的求法，对它们的理解以及在构造 LL(1) 分析表时的使用。递归子程序法中如何体现分析的结果。



# 第4章 自顶向下的语法分析

---

**4.1 语法分析概述**

**4.2 自顶向下的语法分析面临的问题  
与解决方法**

**4.3 预测分析法**

**4.4 递归下降分析法**

**4.5 本章小结**

# 语法分析的功能和位置

■ **语法分析** (syntax analysis) 是编译程序的核心部分，其任务是检查词法分析器输出的单词序列是否是源语言中的句子，亦即是否符合源语言的语法规则。

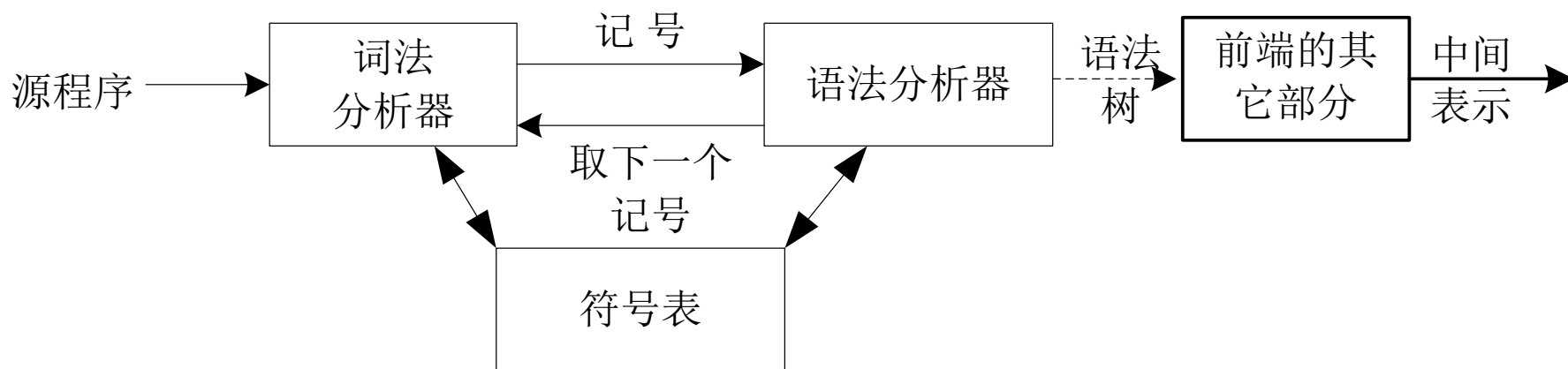


图4.1 语法分析器在编译器中的位置

# 4.1 语法分析概述

用到的是推导技术

- 自顶向下  
Top Down

递归子程序法

从根开始，逐步为某语句构造一棵语法树

预测分析法 (LL(1))

用到的是归约技术

反

- 自底向上  
Bottom Up

算符优先分析法

相反，将一句  
子归约为开始  
符号

LR(0)、SLR(1)、LR(1)、LALR(1)

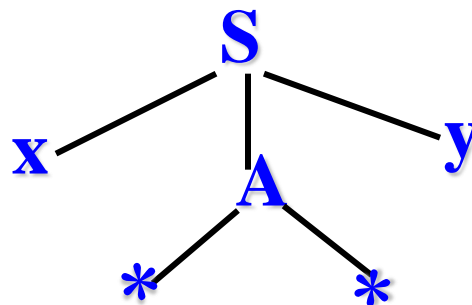
问题：解决确定性问题！

假定文法是压缩的：即删除了单位产生式和无用产生式。

## 4.2 自顶向下的语法分析用到的技术、面临的问题及解决方法

- 自顶向下语法分析的基本思想
  - 从文法的开始符号出发，为输入符号串寻求一个最左推导。
  - 从树根S开始，构造所给输入符号串的语法树
- 例:设有G:  $S \rightarrow xAy$   $A \rightarrow **|*$ , 输入串:  $x**y$

$S \Rightarrow xAy$   
 $\Rightarrow x***y$





## 4.2.1 自顶向下分析用到的技术

---

- **最左推导(Left-most Derivation)**
  - 每次推导都施加在句型的最左边的语法变量上。——与最右归约对应
- **最右推导(Right-most Derivation)**
  - 每次推导都施加在句型的最右边的语法变量上。——与最左归约（规范规约）对应的规范(Canonical)句型



# 语法树——语法分析的结果

## Parse Tree

- 用树的形式表示句型的生成
  - 树根： 开始符号
  - 中间结点： 非终结符
  - 叶结点： 终结符或者非终结符
- 每个推导对应一个中间结点及其儿子——一个二级子树
- 又称为**分析树**(parse tree)、**推导树**(derivation tree)、**派生树**(derivation tree)



# 语法树具有如下特性：

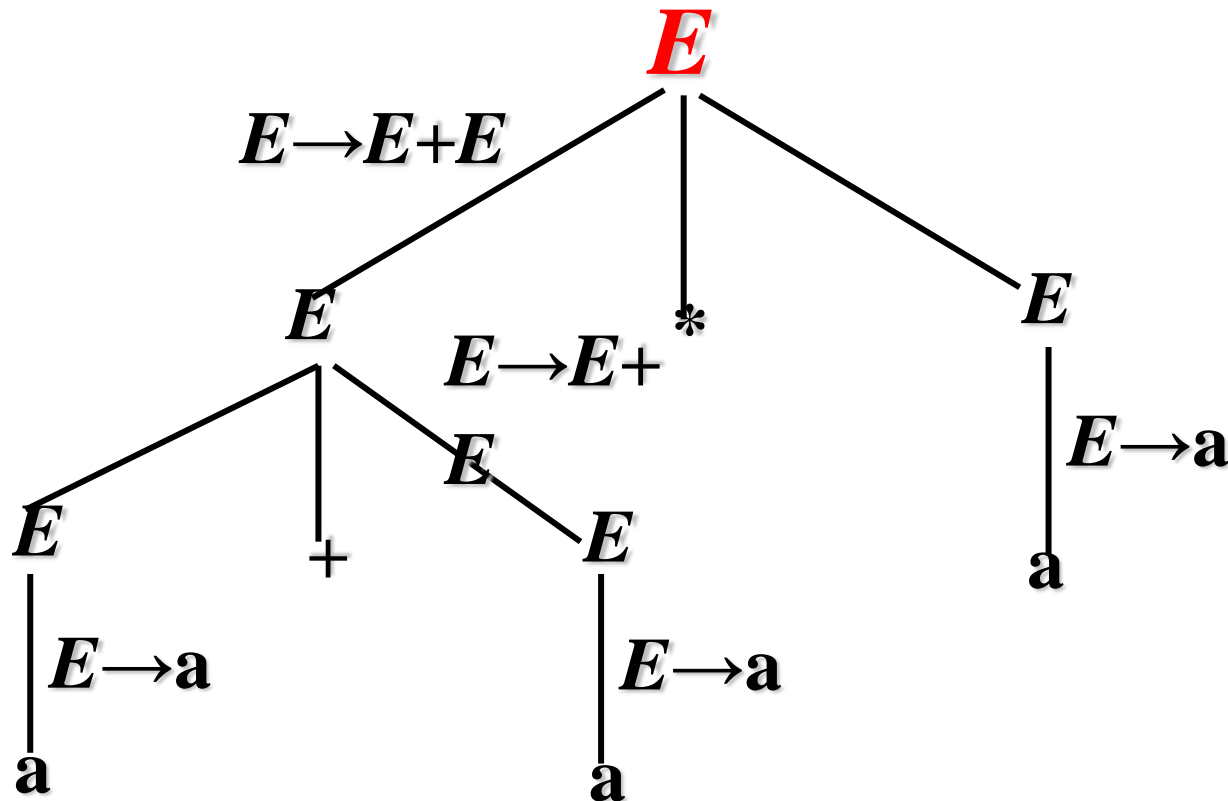
1. 树根标记为开始符号S
2. 每个叶结点由终结符或者  $\epsilon$  标记
3. 每个内结点由一个非终结符标记
4. 如果A是某个内结点的非终结符标记， $A_1, A_2, \dots, A_n$ 是该结点从左到右排列的所有子结点的标记，则 $A \rightarrow A_1 A_2 \dots A_n$ 是一个产生式



# 例 句子结构的表示

(文法  $E \rightarrow E + E \mid E * E \mid (E) \mid a$  )

$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$



一棵树！

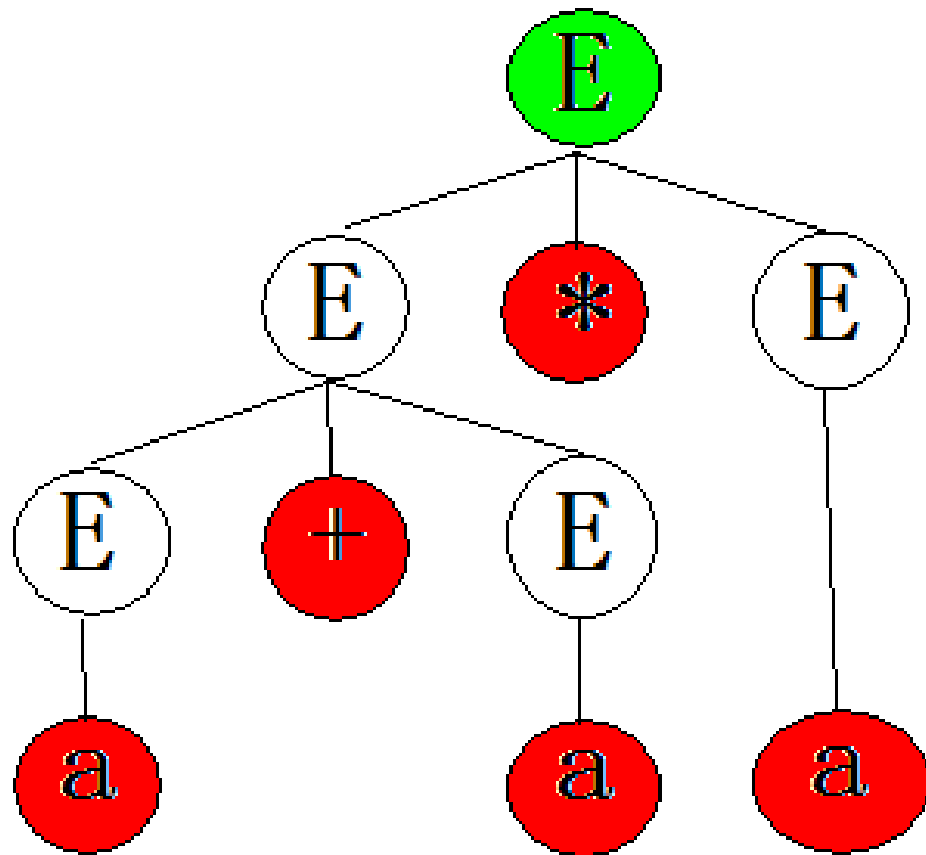
# 关于语法树的几点结论

■ **短语**：一棵**子树**的所有叶子自左至右排列起来形成一个相对于**子树根**的短语。

短语：

**a , a , a ,**

**a+a, a+a\*a**

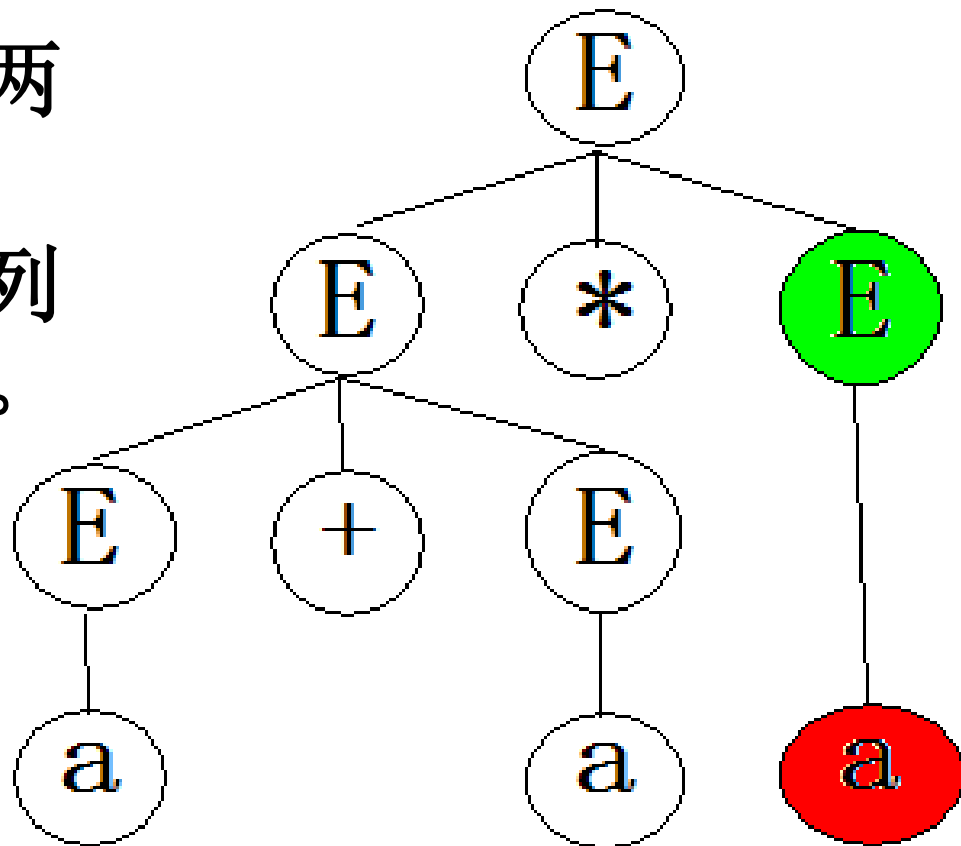


# 关于语法树的几点结论

**直接短语**：仅有父子两代的一棵子树，它的所有叶子自左至右排列起来所形成的符号串。

直接短语：

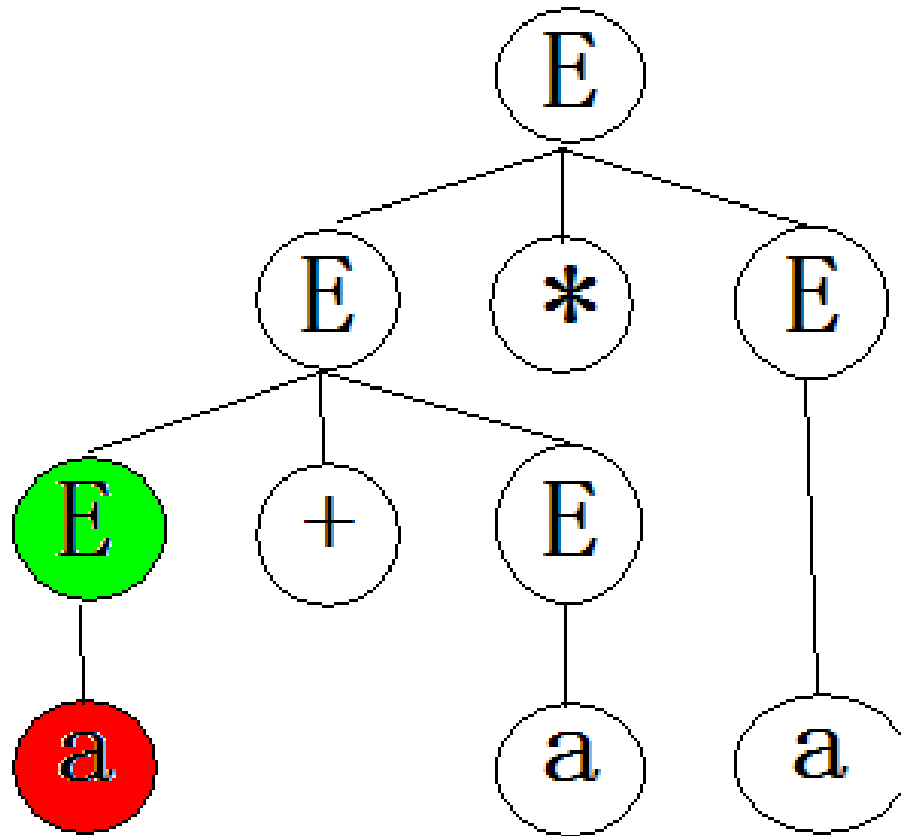
**a , a , a ,**



# 关于语法树的几点结论

■ **句柄**：一个句型的分析树中最左面的直接短语

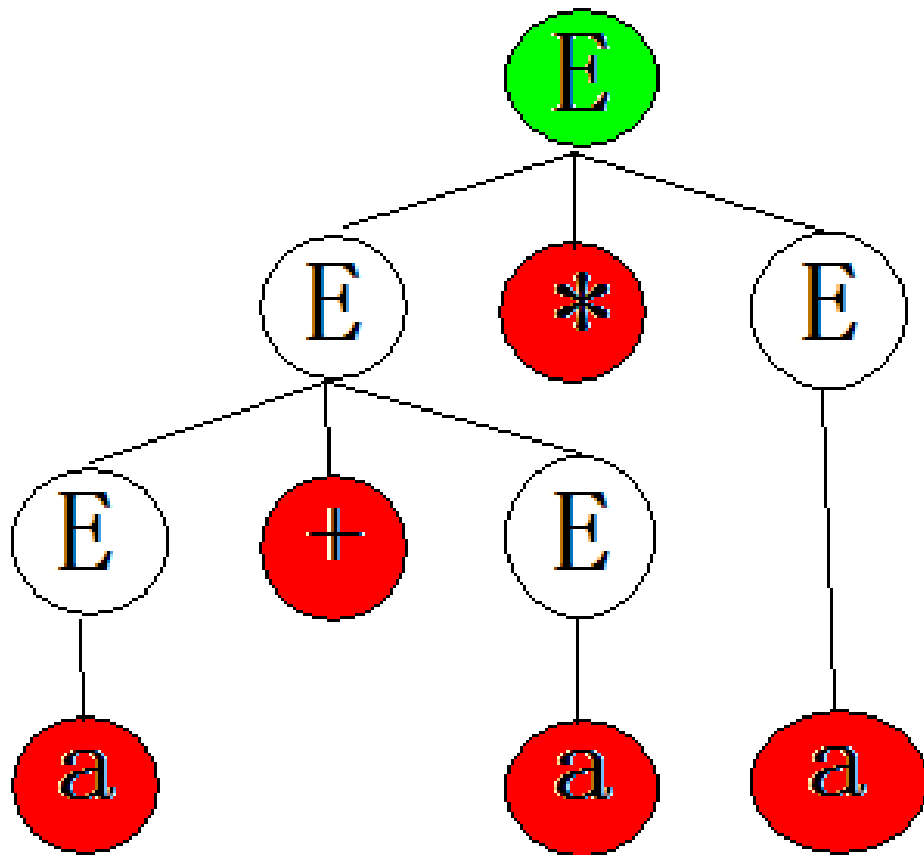
句柄：**a**



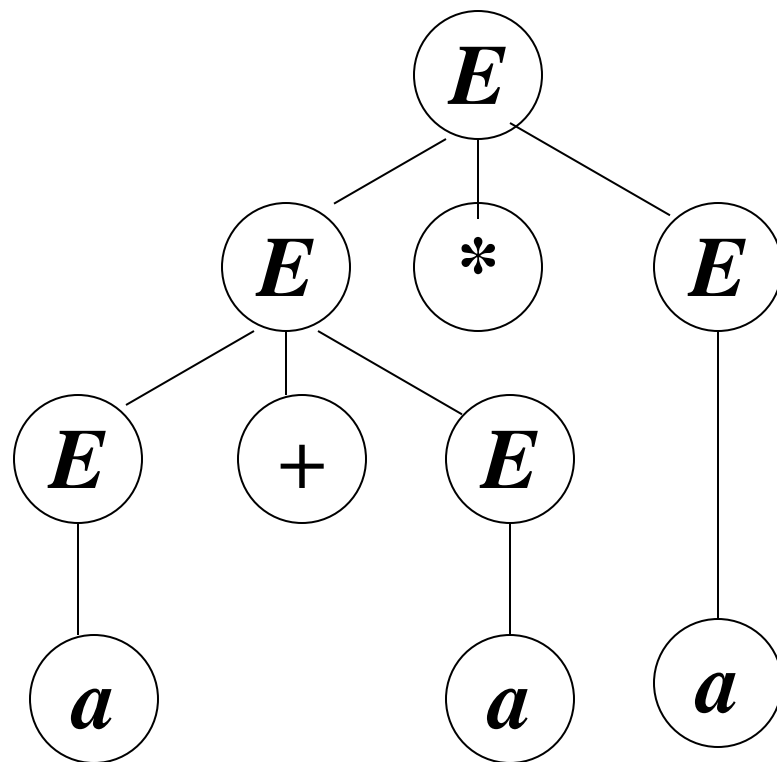
# 关于语法树的几点结论

■ **句子**：语法树的叶点从左到右的排列，刚好是这个文法所生成的语言的一个句子。

句子： **$a + a * a$**

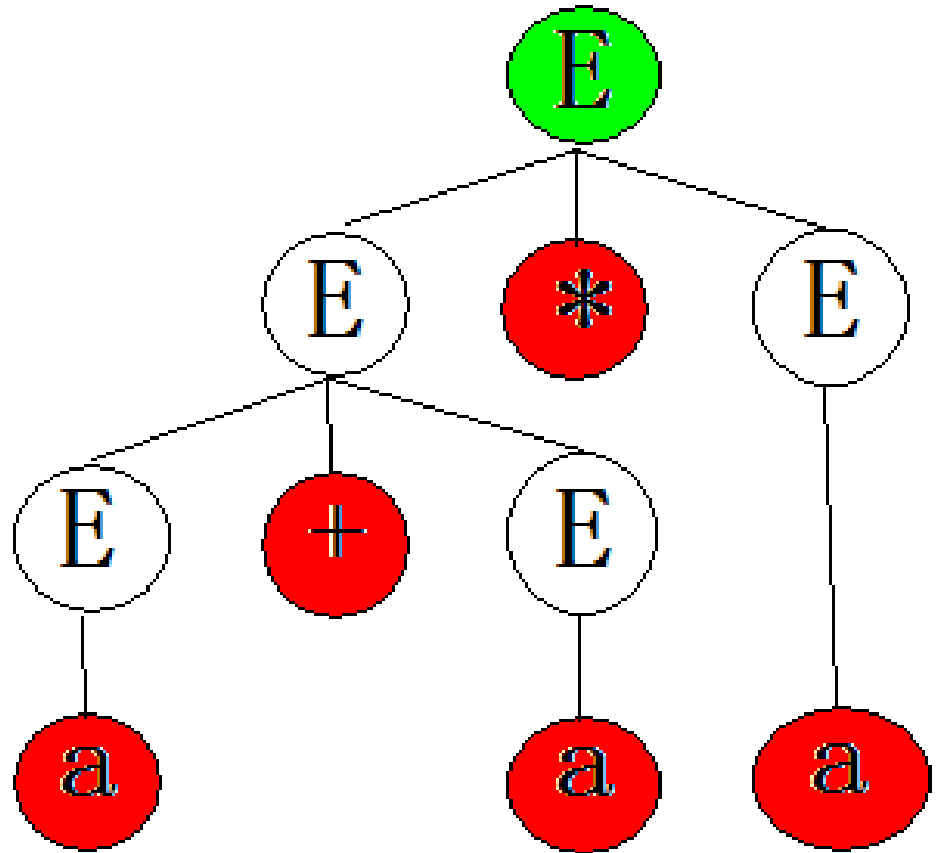


# 关于语法树的几点结论



# 关于语法树的几点结论

短语: **a, a, a,**  
**a+a, a+a\*a**





# 关于语法树的几点结论

---

- 一个文法生成的语言就是它的某个语法树所生成的所有句子的集合
- 为给定的终结字符串（句子）构造一棵语法树的过程称为这个串（句子）的语法分析（parsing）



## 4.2.2 自顶向下分析面临的问题

### 1. 文法的二义性

考虑表达式下面的文法  $G[E]$ , 其产生式如下:

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

对于句子  $a + a * a$ , 有如下两个最左推导:

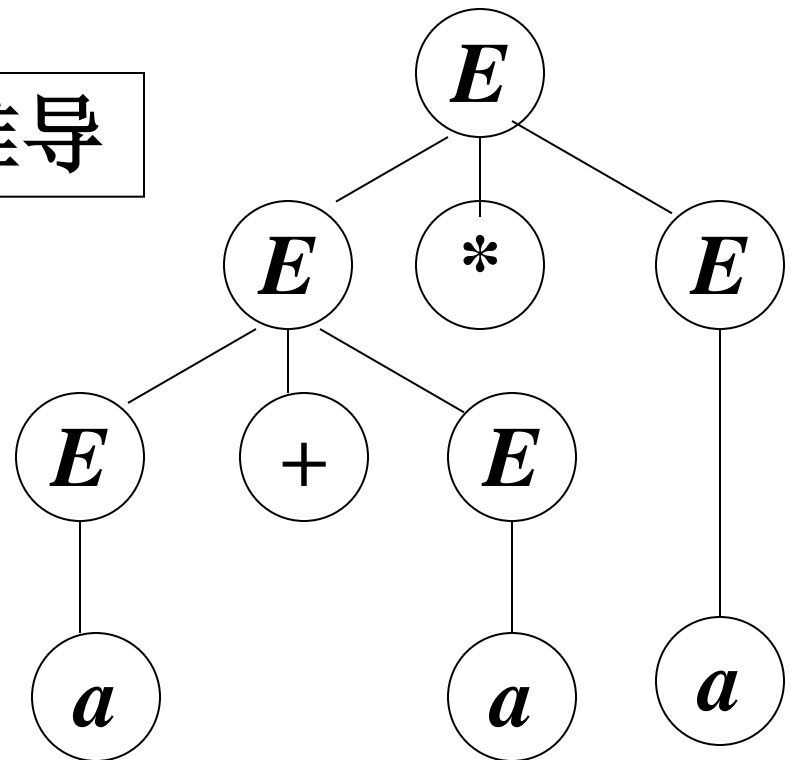
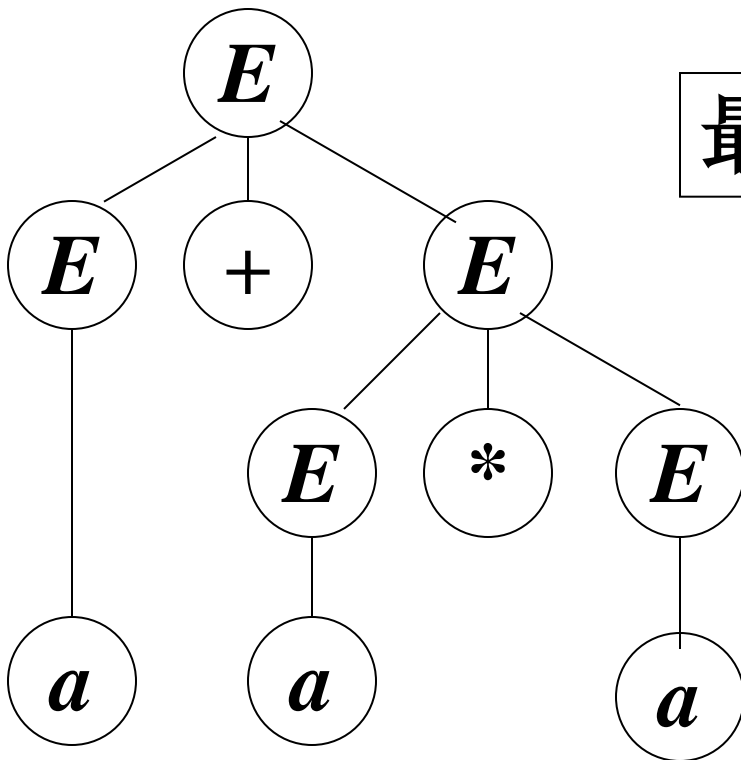
$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \Rightarrow a + a * E \Rightarrow a + a * a$$

$$\begin{aligned}
 E &\Rightarrow E + E \Rightarrow a + E \\
 &\Rightarrow a + E * E \Rightarrow a + a * E \\
 &\Rightarrow a + a * a
 \end{aligned}$$

$$\begin{aligned}
 E &\Rightarrow E * E \Rightarrow E + E * E \\
 &\Rightarrow a + E * E \Rightarrow a + a * E \\
 &\Rightarrow a + a * a
 \end{aligned}$$

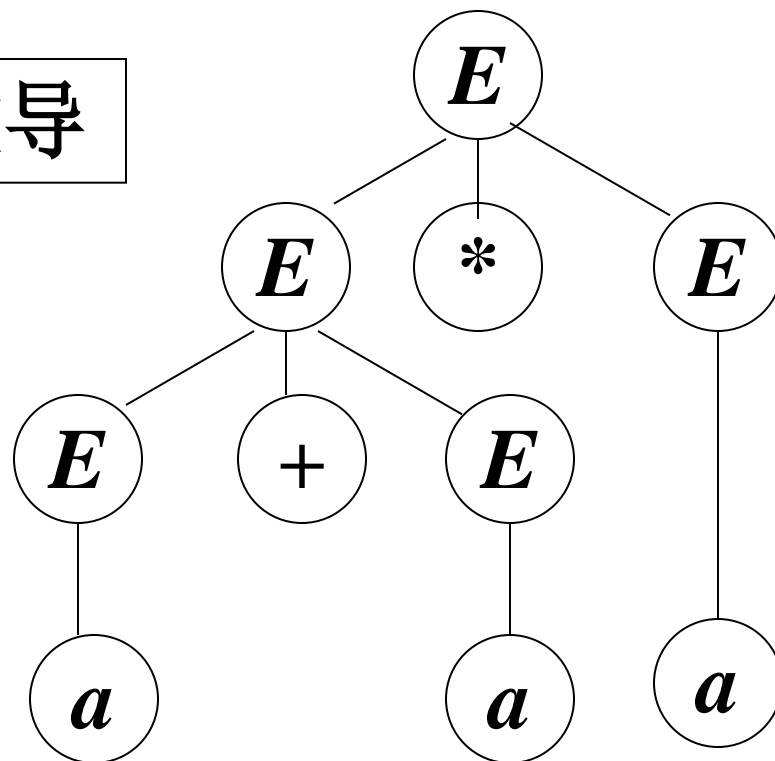
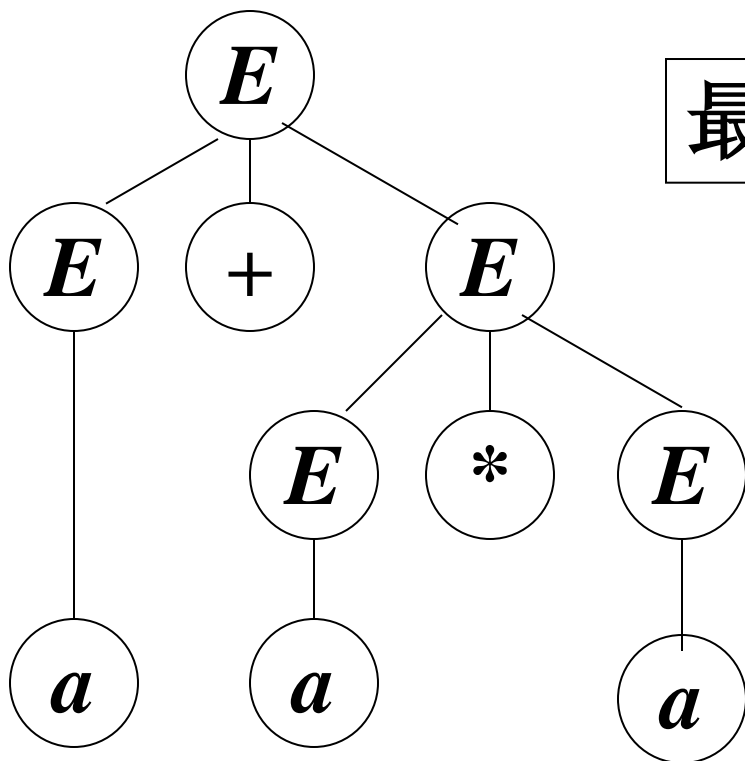
最左推导



$$\begin{aligned}
 E &\Rightarrow E + E \Rightarrow E + E * E \\
 &\Rightarrow E + E * a \Rightarrow E + a * a \\
 &\Rightarrow a + a * a
 \end{aligned}$$

$$\begin{aligned}
 E &\Rightarrow E * E \Rightarrow E * a \\
 &\Rightarrow E + E * a \Rightarrow E + a * a \\
 &\Rightarrow a + a * a
 \end{aligned}$$

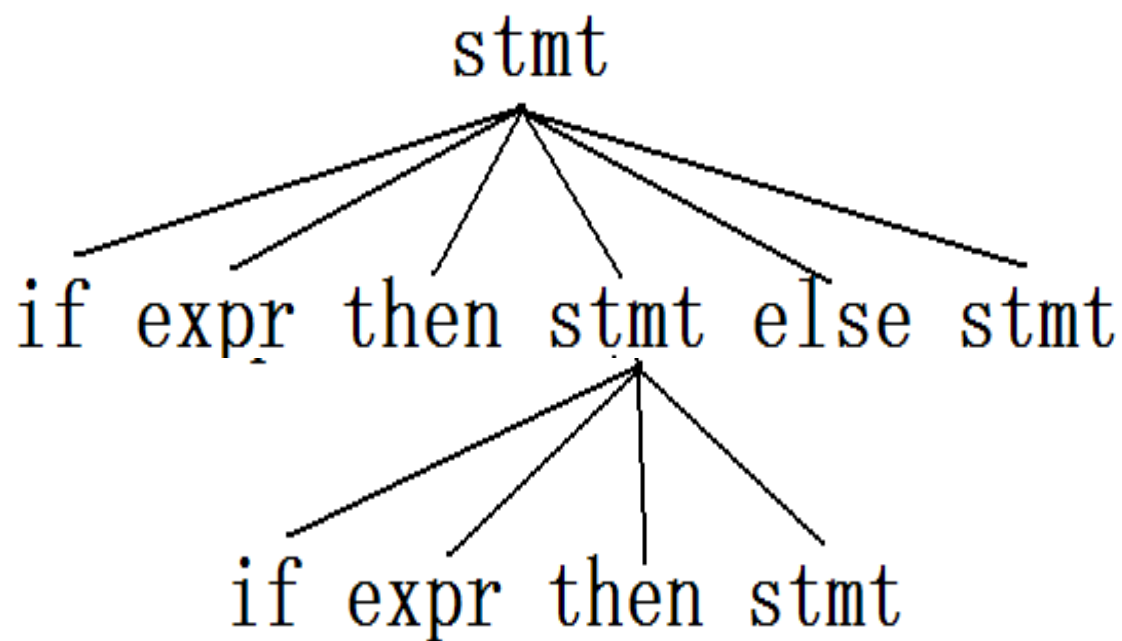
最右推导



# 描述if语句的二义性文法

- 程序语言中的条件语句，经常使用二义性文法描述它：
- $stmt \rightarrow$  if  $expr$  then  $stmt$
- | if  $expr$  then  $stmt$  else  $stmt$
- | other
- 二义性的句子：
- if  $e_1$  then if  $e_2$  then  $s_1$  else  $s_2$
- 很容易构造它的语法树

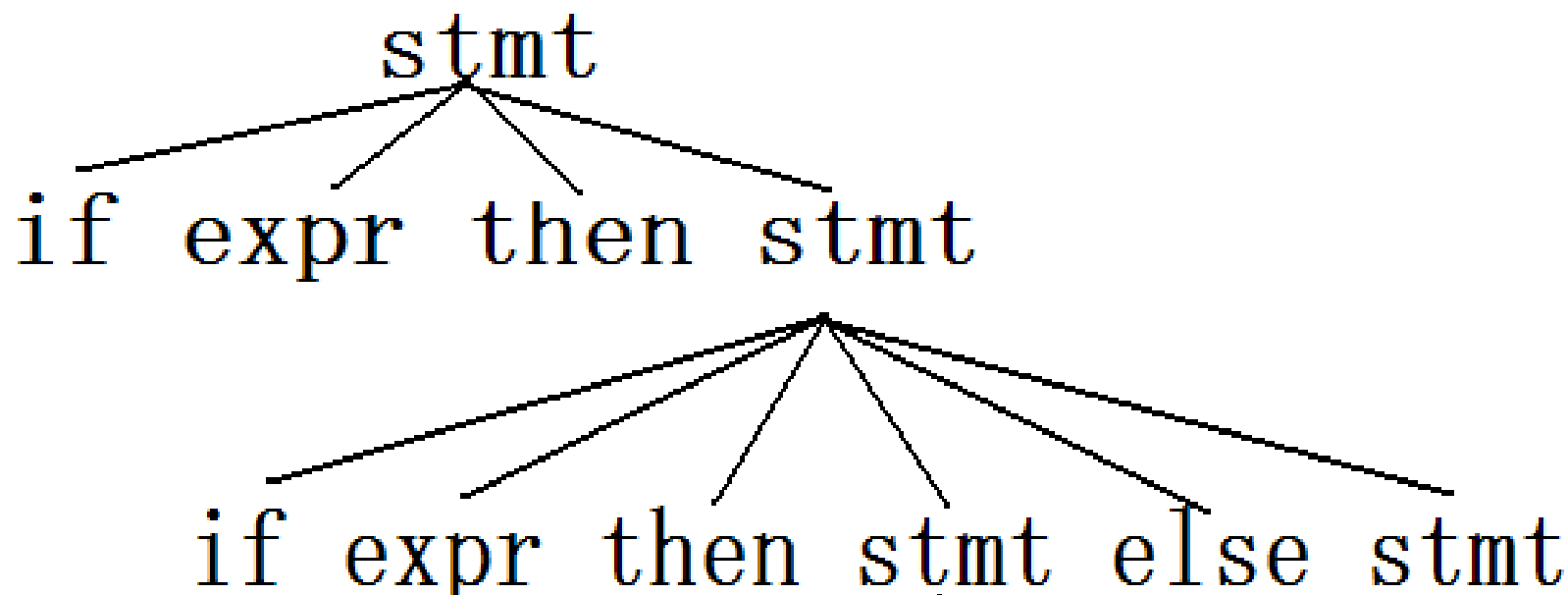
if  $e_1$  then if  $e_2$  then  $s_1$  else  $s_2$ 的语法树





if  $e_1$  then if  $e_2$  then  $s_1$  else  $s_2$ 的语法树

---



# 二义性 (ambiguity)的定义

- 对于文法 $G$ ，如果 $L(G)$ 中存在一个具有两棵或两棵以上分析树的句子，则称 $G$ 是二义性的。也可以等价地说：如果 $L(G)$ 中存在一个具有两个或两个以上最左(或最右)推导的句子，则 $G$ 是二义性文法。
- 如果一个文法 $G$ 是二义性的，假设 $w \in L(G)$ 且 $w$ 存在两个最左推导，则在对 $w$ 进行自顶向下的语法分析时，语法分析程序将无法确定采用 $w$ 的哪个最左推导。

# 4.2.1 自顶向下分析面临的问题

## 2. 回溯问题

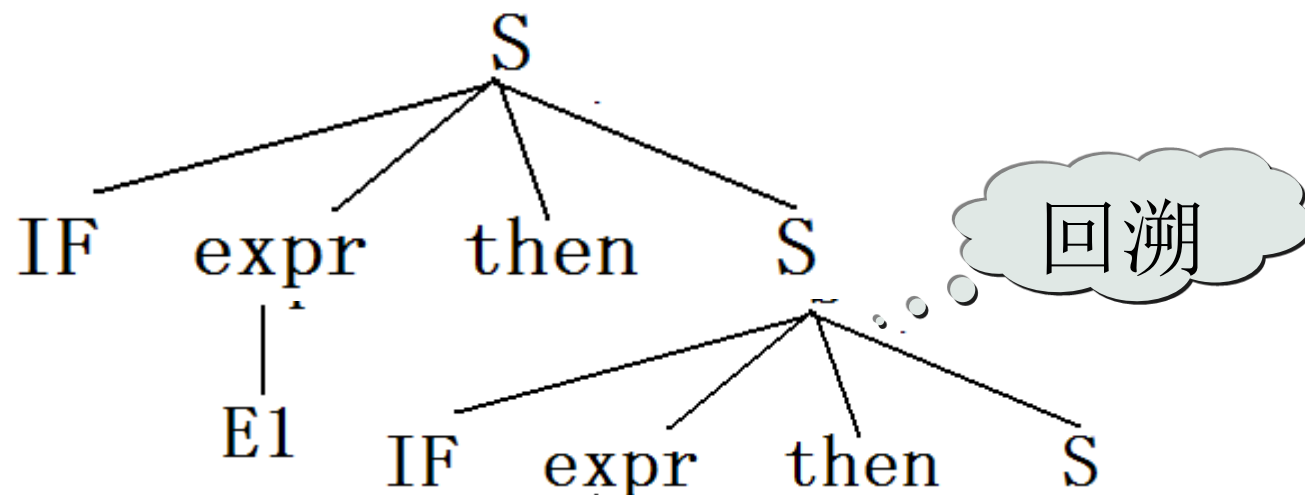
- 文法中每个语法变量 $A$ 的产生式右部称为 $A$ 的候选式，如果 $A$ 有多个候选式存在公共前缀，则自顶向下的语法分析程序将无法根据当前输入符号准确地选择用于推导的产生式，只能试探。
- 当试探不成功时就需要退回到上一步推导，看 $A$ 是否还有其它的候选式，这就是回溯(backtracking)。



## 4.2.2 自顶向下分析面临的问题

文法:  $S \rightarrow$  if  $expr$  then  $S$   
          | if  $expr$  then  $S$  else  $S$   
          | other

为句子if **E1** then **S1** else **S2**构造一棵语法树



## 4.2.2 自顶向下分析面临的问题

- 造成这种情况的原因是产生式具有相同的首符号，

- $$S \rightarrow \begin{array}{l} \text{if } \textit{expr} \text{ then } S \\ \text{if } \textit{expr} \text{ then } S \text{ else } S \\ \text{other} \end{array}$$

对于句子 **if E1 then S1 else S2** 来说

从而导致不清楚该用哪个来替换非终结符



## 4.2.2 自顶向下分析面临的问题

---

可通过**改写**产生式来**推迟**这种决定，直到**看见**足够多的输入符号，可以作出正确选择为止

- 具体将采用提取左因子的方法来改造文法，以便减少推导过程中回溯现象的发生

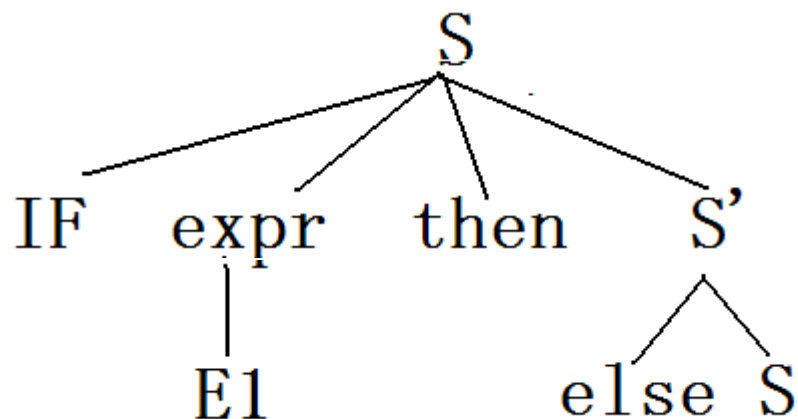
## 4.2.2 自顶向下分析面临的问题

- 上例文法可改为:

- $S \rightarrow \text{if } \textit{expr} \text{ then } S \ S' / S$

- $S' \rightarrow \text{else } S \ / \ \epsilon$

句子if E1 then S1 else S2具有唯一的语法树



## 4.2.2 自顶向下分析面临的问题

### 3. 左递归

假设 $A$ 是文法 $G$ 的某个语法变量，如果存在推导

$A \Rightarrow^+ \alpha A \beta$ ，则称文法 $G$ 是递归的，

当 $\alpha = \varepsilon$ 时，即 $A \Rightarrow^+ A \beta$ 称之为左递归；

如果 $A \Rightarrow^+ \alpha A \beta$ 至少需要两步推导，则称文法 $G$ 是间接递归的，当 $\alpha = \varepsilon$ 时称之为间接左递归

如果文法 $G$ 中存在形如 $A \rightarrow \alpha A \beta$ 的产生式，则称文法 $G$ 是直接递归的，当 $\alpha = \varepsilon$ 时称之为直接左递归。

例：下面是描述算术表达式的算法

■  $E \rightarrow E + T \mid T$

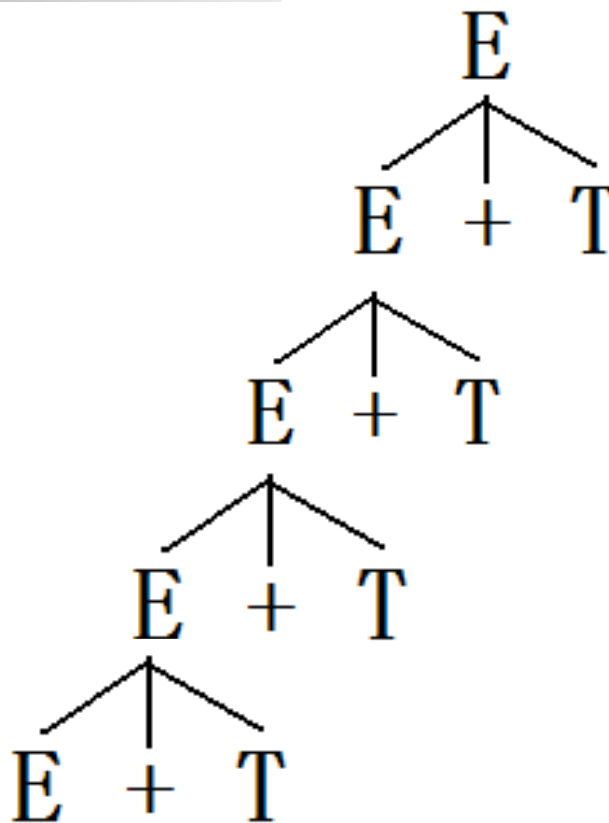
■  $T \rightarrow T * F \mid F$

■  $F \rightarrow (E) \mid \text{id}$

■ 为句子 **id\*id+id**

■ 构造分析树

■ 左递归会让分析进入无限循环之中





## 4.2.3 对上下文无关文法的改造

### 1. 消除二义性

- 改造的方法就是通过引入新的语法变量等，使文法含有更多的信息。其实，许多二义性文法是由于概念不清，即语法变量的定义不明确导致的，此时通过引入新的语法变量即可消除文法的二义性。

# 描述算术表达式的二义文法

- 文法  $G[E]$ :
- $E \rightarrow E+E \mid E-E \mid E * E \mid E/E \mid (E) \mid a$
- 造成二义性的原因是：文法中没有体现出结合率和优先级

解决办法：改造文法，引入新的文法变量

- $G_{exp}$ :  
$$E \rightarrow E+T \mid E-T \mid T$$
$$T \rightarrow T * F \mid T / F \mid F$$
$$F \rightarrow a \mid (E)$$





# 描述IF语句的二义文法

- $\langle stmt \rangle \rightarrow \text{if } \langle expr \rangle \text{ then } \langle stmt \rangle$
- |  $\text{if } \langle expr \rangle \text{ then } \langle stmt \rangle \text{ else } \langle stmt \rangle$
- |  $\text{other}$  (4.7)
- 根据if语句中else与then配对情况将其分为配对的语句和不配对的语句两类。上述if语句的文法没有对这两个不同的概念加以区分，只是简单地将它们都定义为 $\langle stmt \rangle$ ，从而导致该文法是二义性的。

## 4.2.3 对上下文无关文法的改造

引入语法变量 *<unmatched\_stmt>* 来表示不配对语句, *<matched\_stmt>* 表示配对语句

- $\langle stmt \rangle \rightarrow \langle matched\_stmt \rangle$   
                  |  $\langle unmatched\_stmt \rangle$
- $\langle matched\_stmt \rangle \rightarrow \text{if } \langle expr \rangle \text{ then}$   
                   $\langle matched\_stmt \rangle \text{ else } \langle matched\_stmt \rangle$   
                  | other
- $\langle unmatched\_stmt \rangle \rightarrow \text{if } \langle expr \rangle \text{ then } \langle stmt \rangle$   
                  |  $\text{if } \langle expr \rangle \text{ then } \langle matched\_stmt \rangle \text{ else}$   
                   $\langle unmatched\_stmt \rangle$



## 4.2.3 对上下文无关文法的改造

### 2. 消除左递归

消除简单左递归的方法：

对于含有左递归的产生式  $A \rightarrow A\alpha \mid \beta$

可用下面的非左递归的产生式 代替：

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$



## 4.2.3 对上下文无关文法的改造

---

- $E \rightarrow E + T \mid T$
- $T \rightarrow T * F \mid F$
- $F \rightarrow (E) \mid \text{id}$  替换为:
- $E \rightarrow TE'$
- $E' \rightarrow +TE' \mid \varepsilon$
- $T \rightarrow FT'$
- $T' \rightarrow *FT' \mid \varepsilon$
- $F \rightarrow (E) \mid \text{id}$

## 4.2.3 对上下文无关文法的改造

对于一般情况而言，若某一文法G的产生式具有如下形式：

$$A \rightarrow A \alpha_1 \mid A \alpha_2 \mid \dots \mid A \alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

则可用如下方法消除左递归：

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$$

很容易证明改造前后的文法是等价的

## 4.2.3 对上下文无关文法的改造

算法4.1 消除左递归。

输入：不含循环推导和 $\varepsilon$ -产生式的文法 $G$ ；

输出：与 $G$ 等价的无左递归文法；

步骤：

1. 将 $G$ 的所有语法变量排序(编号)，假设排序后的语法变量记为 $A_1, A_2, \dots, A_n$ ；
2. for  $i \leftarrow 1$  to  $n$  {
3.     for  $j \leftarrow 1$  to  $i-1$  {
4.         用产生式 $A_i \rightarrow \alpha_1 \beta | \alpha_2 \beta | \dots | \alpha_k \beta$ 代替每个形如 $A_i \rightarrow A_j \beta$ 的产生式，
- 其中， $A_j \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$ 是所有的当前 $A_j$ 产生式；
5.     }
6.     消除 $A_i$ 产生式中的所有直接左递归
7. }

## 4.2.3对上下文无关文法的改造

### ■ 3. 提取左因子

- 对每个语法变量 $A$ ，找出它的两个或更多候选式的最长公共前缀 $\alpha$ 。如果 $\alpha \neq \varepsilon$ ，则用下面的产生式替换所有的 $A$ 产生式 $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n | \gamma_1 | \gamma_2 | \dots | \gamma_n$ ，其中 $\gamma_1, \gamma_2, \dots, \gamma_n$ 表示所有不以 $\alpha$ 开头的候选式：

$$A \rightarrow \alpha A' | \gamma_1 | \gamma_2 | \dots | \gamma_n$$

$$A' \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$$

- 其中， $A'$ 是新引入的语法变量。反复应用上述变换，直到任意语法变量都没有两个候选式具有公共前缀为止。请读者自行给出这个变换的算法。

- 
- 例：文法**G** (**P**) :

$$\mathbf{P} \rightarrow (\mathbf{Q}) \mid \mathbf{aP} \mid \mathbf{a}$$

- $\mathbf{Q} \rightarrow \mathbf{Q}, \mathbf{P} \mid \mathbf{P}$

- 消除左递归、消除回溯

- 解：消除左递归

- $\mathbf{Q} \rightarrow \mathbf{PQ}'$

- $\mathbf{Q}' \rightarrow , \mathbf{PQ}' \mid \varepsilon$

- 消除回溯

- $\mathbf{P} \rightarrow (\mathbf{Q}) \mid \mathbf{aP}'$

- $\mathbf{P}' \rightarrow \mathbf{P} \mid \varepsilon$





## 4.2.4 LL(1)文法

问题:

什么样的文法对其句子才能进行确定的自顶向下分析?

- 确定的自顶向下分析首先从文法的开始符号出发, 每一步推导都根据当前句型的最左语法变量 $A$ 和当前输入符号 $a$ , 选择 $A$ 的某个候选式 $\alpha$ 来替换 $A$ , 并使得从 $\alpha$ 推导出的第一个终结符恰好是 $a$ 。
- 当 $A$ 有多个候选式时, 当前选中的候选式必须是惟一的。
- 第一个终结符是指符号串的第一个符号, 并且是终结符号, 可以称为首终结符号。在自顶向下的分析中, 它对选取候选式具有重要的作用。为此引入首符号集的概念。



# FIRST集的定义

1. 假设 $\alpha$ 是文法 $G=(V, T, P, S)$ 的符号串, 即 $\alpha \in (V \cup T)^*$ ,  
从 $\alpha$ 推导出的串的首符号集记作**FIRST**( $\alpha$ ):

**FIRST**( $\alpha$ )= $\{a | \alpha \xRightarrow{*} a\beta, a \in T, \beta \in (V \cup T)^*\}$ 。

2. 如果 $\alpha = \varepsilon$ , 则 $\varepsilon \in \text{FIRST}(\alpha)$ 。

3. 如果文法 $G$ 中的所有 $A$ 产生式为 $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_m$ ,

且 $\varepsilon \notin \text{FIRST}(\alpha_1) \cup \text{FIRST}(\alpha_2) \cup \dots \cup \text{FIRST}(\alpha_m)$

且对 $\forall i, j, 1 \leq i, j \leq m; i \neq j$ , 均有

$\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \emptyset$ 成立, 则可以对 $G$ 的句子  
进行确定的自顶向下分析



# FOLLOW集的定义

如果存在 $A \rightarrow \varepsilon$ 这样的产生式，则需定义FOLLOW(A)

$\forall A \in V$ 定义A的后续符号集为：

1.  $\text{FOLLOW}(A) = \{a \mid S \xRightarrow{*} \alpha A a \beta, a \in T, \alpha, \beta \in (V \cup T)^*\}$
2. 如果A是某个句型的最右符号，则将结束符#添加到FOLLOW(A)中
3. 如果 $\alpha_j \xRightarrow{*} \varepsilon$ ，则如果对 $\forall i (1 \leq i \leq m; i \neq j)$ ， $\text{FIRST}(\alpha_i) \cap \text{FOLLOW}(A) = \emptyset$ 均成立，则可以对G的句子进行确定的自顶向下分析

**First和Follow  
的用处**

例:  $A \rightarrow aA$

$\text{FIRST}(aA) = \{a\}$

$A \rightarrow bA$

$\text{FIRST}(bA) = \{b\}$

$A \rightarrow cAB$

$\text{FIRST}(cA) = \{c\}$

$A \rightarrow \epsilon$

$\text{FIRST}(\epsilon) = \{\epsilon\}$

$B \rightarrow dC$

$\text{FOLLOW}(A) = \{d\}$

.....

对句子abcd....进行分析

$A \Rightarrow aA \Rightarrow abA \Rightarrow abcAB \Rightarrow abcB \Rightarrow abcdC$



## 4.2.4 LL(1)文法

如果 $G$ 的任意两个具有相同左部的产生式 $A \rightarrow \alpha | \beta$ 满足下列条件:

1. 如果 $\alpha$ 、 $\beta$ 均不能推导出 $\varepsilon$ , 则 $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$ ;
2.  $\alpha$ 和 $\beta$ 至多有一个能推导出 $\varepsilon$ ;
3. 如果 $\beta \xrightarrow{*} \varepsilon$ , 则 $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$

则称 $G$ 为LL(1)文法。

第一个 $L$ 代表从左向右扫描输入符号串, 第二个 $L$ 代表产生最左推导, 1代表在分析过程中执行每步推导都要向前查看一个输入符号

# 求FIRST( $X$ )集的算法

算法4.2 计算FIRST( $X$ )。

输入：文法 $G=(V, T, P, S)$ ,  $X \in (V \cup T)$ ;

输出：FIRST( $X$ );

# 求FIRST(X)集的算法



步骤:

1.  $\text{FIRST}(X) = \emptyset$ ;
  2. if  $(X \in T)$  then  $\text{FIRST}(X) := \{X\}$  ;
  3. if  $X \in V$  then  
begin
  4. if  $(X \rightarrow \varepsilon \notin P)$  then  $\text{FIRST}(X) := \text{FIRST}(X) \cup \{a | X \rightarrow a \dots \in P \text{ and } a \in T\}$ ;
  5. if  $(X \rightarrow \varepsilon \in P)$  then  $\text{FIRST}(X) := \text{FIRST}(X) \cup \{\varepsilon\}$
  - end
  6. 对  $\forall X \in V$ , 重复如下的过程7-10, 直到所有FIRST集不变为止。
  7. if  $(X \rightarrow Y \dots \in P \text{ and } Y \in V \text{ and } Y \rightarrow \varepsilon \notin P)$   
then  $\text{FIRST}(X) := \text{FIRST}(X) \cup (\text{FIRST}(Y) - \{\varepsilon\})$ ;
  8. if  $(X \rightarrow Y_1 \dots Y_n \in P \text{ and } Y_1 \dots Y_{i-1} \xrightarrow{*} \varepsilon)$  then
  9.  $\text{FIRST}(X) := \text{FIRST}(X) \cup (\text{FIRST}(Y_i) - \{\varepsilon\})$ ;
  10. if  $Y_1 \dots Y_n \xrightarrow{*} \varepsilon$  then  $\text{FIRST}(X) := \text{FIRST}(X) \cup \{\varepsilon\}$ ;
- $\Rightarrow$



# LL(1)文法的判定

算法4.3 计算FIRST( $\alpha$ )。

输入：文法 $G=(V, T, P, S)$ ,  $\alpha \in (V \cup T)^*$ ,  $\alpha = X_1 \dots X_n$ ;

输出：FIRST( $\alpha$ );

步骤：

1. 计算FIRST( $X_1$ );
2. FIRST( $\alpha$ ) := FIRST( $X_1$ ) -  $\{\epsilon\}$ ;
3.  $k := 1$ ;
4. while ( $\epsilon \in \text{FIRST}(X_k)$  and  $k < n$ ) do begin
5.     FIRST( $\alpha$ ) := FIRST( $\alpha$ )  $\cup$  (FIRST( $X_{k+1}$ ) -  $\{\epsilon\}$ );
6.      $k := k + 1$  end
7. if ( $k = n$  and  $\epsilon \in \text{FIRST}(X_k)$ ) then FIRST( $\alpha$ ) := FIRST( $\alpha$ )  $\cup$   $\{\epsilon\}$ ;



# 例 表达式文法的语法符号的FIRST 集

$\text{FIRST}(F) = \{ (, \text{id} \}$

$\text{FIRST}(T) = \text{FIRST}(F) = \{ (, \text{id} \}$

$\text{FIRST}(E) = \text{FIRST}(T) = \{ (, \text{id} \}$

$\text{FIRST}(E') = \{ +, \epsilon \}$

$\text{FIRST}(T') = \{ *, \epsilon \}$

$\text{FIRST}(+) = \{ + \}, \text{FIRST}(*) = \{ * \}$

$\text{FIRST}( ( ) = \{ ( \}$

$\text{FIRST}( ) ) = \{ ) \}$

$\text{FIRST}(\text{id}) = \{ \text{id} \}$

$E \rightarrow TE'$

$E' \rightarrow +TE' | \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' | \epsilon$

$F \rightarrow (E) | \text{id}$



# LL(1)文法的判定

算法4.4 计算FOLLOW集。

输入：文法 $G=(V, T, P, S)$ ,  $A \in V$ ;

输出：FOLLOW( $A$ );

步骤：

1. 对 $\forall X \in V$ , FOLLOW( $S$ ) :=  $\emptyset$ ;
2. FOLLOW( $S$ ) :=  $\{\#\}$ , #为句子的结束符;
3. 对 $\forall X \in V$ , 重复下面的第4步到第5步, 直到所有 FOLLOW集不变为止。
4. 若 $A \rightarrow \alpha B \beta \in P$ , 则  
FOLLOW( $B$ ) := FOLLOW( $B$ )  $\cup$  FIRST( $\beta$ ) -  $\{\varepsilon\}$ ;
5. 若 $A \rightarrow \alpha B$  或  $A \rightarrow \alpha B \beta \in P$ , 且 $\beta \xrightarrow{\varepsilon} \varepsilon$ ,  $A \neq B$ , 则  
FOLLOW( $B$ ) := FOLLOW( $B$ )  $\cup$  FOLLOW( $A$ );

# 例 表达式文法的语法变量的 FOLLOW 集

$E \rightarrow TE' \quad E' \rightarrow +TE' | \epsilon$

$T \rightarrow FT' \quad T' \rightarrow *FT' | \epsilon$

$F \rightarrow (E) | id$

$FIRST(F) = \{ (, id \}$

$FIRST(T) = FIRST(F) = \{ (, id \}$

$FIRST(E) = FIRST(T) = \{ (, id \}$

$FIRST(E') = \{ +, \epsilon \}$

$FIRST(T') = \{ *, \epsilon \}$

$FOLLOW(E) = \{ \#, ) \}$

$FOLLOW(E') = FOLLOW(E) = \{ \#, ) \}$

$FOLLOW(T) = FIRST(E') \cup FOLLOW(E) \cup FOLLOW(E') = \{ +, ), \# \}$

$FOLLOW(T') = FOLLOW(T) = \{ +, ), \# \}$

$FOLLOW(F) = FIRST(T') \cup FOLLOW(T) \cup FOLLOW(T') = \{ *, +, ), \# \}$



# 复习

---

- **1.问题**
- 二义性
- 回溯
- 左递归

# 复习

- 2.解决方法
- 二义性：通过具体的语义来消除
- 回溯：提取左因子
  - $A \rightarrow \alpha\beta_1|\alpha\beta_2|\dots|\alpha\beta_n|\gamma_1|\gamma_2|\dots|\gamma_m$ ，其中 $\gamma_1$ ,  $\gamma_2$ , ...,  $\gamma_m$ 表示所有不以 $\alpha$ 开头的候选式：

$$A \rightarrow \alpha A' |\gamma_1|\gamma_2|\dots|\gamma_m$$

$$A' \rightarrow \beta_1|\beta_2|\dots|\beta_n$$



# 复习

---

左递归：转换为右递归

$$A \rightarrow A \alpha_1 \mid A \alpha_2 \mid \dots \mid A \alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

则可用如下方法消除左递归：

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$$



# 复习

---

- **3. first集和follow集**

- 定义:

1)  $FIRST(\alpha) = \{a | \alpha \Rightarrow a\beta, a \in T, \beta \in (V \cup T)^*\}$ 。

2) 如果  $\alpha \Rightarrow \varepsilon$ , 则  $\varepsilon \in FIRST(\alpha)$ 。



# 复习--求FIRST集算法：

---

步骤：

1. if  $(X \in T)$  then  $\text{FIRST}(X) := \{X\}$  ;

2. if  $X \in V$  then

if  $(X \rightarrow Y_1 \dots Y_n \in P)$  then

$\text{FIRST}(X) := \text{FIRST}(X) \cup \text{FIRST}(Y_1 \dots Y_n)$





# 复习--求FOLLOW集算法:

---

步骤:

1.  $\text{FOLLOW}(S) := \{\#\}$ , #为句子的结束符;

2. 若  $A \rightarrow \alpha B \beta \in P$   $\beta \neq \varepsilon$

$\text{FOLLOW}(B) := \text{FOLLOW}(B) \cup \text{FIRST}(\beta) - \{\varepsilon\}$ ;

3. 若  $A \rightarrow \alpha B$  或  $A \rightarrow \alpha B \beta \in P$ , 且  $\beta \Rightarrow \varepsilon$ ,  $A \neq B$ , 则

$\text{FOLLOW}(B) := \text{FOLLOW}(B) \cup \text{FOLLOW}(A)$ ;



# 复习

---

**FIRST集与FOLLOW集的用法:**

**FIRST集用来帮助我们在分析时选择用来做分析的非空产生式**

**FOLLOW集用来帮助我们在分析时选择用空产生式来做分析**

# 复习 LL(1)文法

1.不含左递归

2.不含回溯

如果 $G$ 的任意两个具有相同左部的产生式

$A \rightarrow \alpha | \beta$ 满足下列条件:

1) . 如果 $\alpha$ 、 $\beta$ 均不能推导出 $\varepsilon$ , 则

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset;$$

2).  $\alpha$ 和 $\beta$ 至多有一个能推导出 $\varepsilon$ ;假设 $\beta \Rightarrow \varepsilon$ , 则

$$\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \emptyset$$

只有LL(1)文法, 才可以实现确定的自顶向下语法分析



## 练习：消除左递归、提取公因子

---

$Z \rightarrow A$

$A \rightarrow aB | aC | Ad | Ae$

$B \rightarrow bBC | f$

$C \rightarrow c$

消除左递归：

$A \rightarrow aBA' | aCA'$

$A' \rightarrow dA' | eA' | \epsilon$



---

再提取公因子:

$A' \rightarrow aA''$

$A'' \rightarrow BA'$

$A'' \rightarrow CA'$

消除左递归:

$A \rightarrow aBA' \mid aCA'$

$A' \rightarrow dA' \mid eA' \mid \epsilon$



改造后文法为:

1、  $Z \rightarrow A$

6、  $A' \rightarrow eA'$

2、  $A \rightarrow aA''$

7、  $A' \rightarrow \epsilon$

3、  $A'' \rightarrow BA'$

8、  $B \rightarrow bBC$

4、  $A'' \rightarrow CA'$

9、  $B \rightarrow f$

5、  $A' \rightarrow dA'$

10、  $C \rightarrow c$



改造后文法为:

$Z \rightarrow A$

$A \rightarrow aA''$

$A'' \rightarrow BA'$

$A'' \rightarrow CA'$

$A' \rightarrow dA' \mid eA' \mid \epsilon$

$B \rightarrow bBC \mid f$

$C \rightarrow c$

$\text{FIRST}(Z) = \{a\}$

$\text{FIRST}(A) = \{a\}$

$\text{FIRST}(A'') = \{b, f, c\}$

$\text{FIRST}(A') = \{d, e, \epsilon\}$

$\text{FIRST}(B) = \{b, f\}$

$\text{FIRST}(C) = \{c\}$



改造后文法为:

$Z \rightarrow A$

$A \rightarrow aA''$

$A'' \rightarrow BA'$

$A'' \rightarrow CA'$

$A' \rightarrow dA' \mid eA' \mid \epsilon$

$B \rightarrow bBC \mid f$

$C \rightarrow c$

$FOLLOW(Z) = \{\#\}$

$FOLLOW(A) = \{\#\}$

$FOLLOW(A'') = \{\#\}$

$FOLLOW(A') = \{\#\}$

$FOLLOW(B) = \{c, d, e, \#\}$

$FOLLOW(C) = \{c, d, e, \#\}$





# 是LL(1)吗

---

**$\text{FIRST}(dA') \cap \text{FIRST}(eA') = \emptyset$ ;**

**且  $\text{FIRST}(dA') \cap \text{FOLLOW}(A) \neq \emptyset$  和**

**$\text{FIRST}(eA') \cap \text{FOLLOW}(A) \neq \emptyset$  ;**

**又  $\text{FIRST}(bBC) \cap \text{FIRST}(f) = \emptyset$ ;**

**所以改造后文法是LL(1)的**



## 练习：消除左递归、提取公因子

**$Z \rightarrow A$**

---

**$A \rightarrow aB|aC|Ad|Ae$**

**$B \rightarrow bBC|f$**

**$C \rightarrow c$**

先提左因子

**$A \rightarrow aA'|AA''$**

**$A' \rightarrow B|C$**

**$A'' \rightarrow d|e$**



## 先提左因子

---

$A \rightarrow aA' | AA''$

$A' \rightarrow B | C$

$A'' \rightarrow d | e$

## 消除左递归

$A \rightarrow aA' A'''$

$A''' \rightarrow A'' A''' | \epsilon$



改造后文法:

$Z \rightarrow A$

$A \rightarrow aA' A''$

$A' \rightarrow B|C$

$A'' \rightarrow d|e$

$A''' \rightarrow A'' A''' | \epsilon$

$B \rightarrow bBC|f$

$C \rightarrow c$

$\text{First}(Z) = \{a\}$

$\text{First}(A) = \{a\}$

$\text{First}(A') = \{b, f, c\}$

$\text{First}(A'') = \{d, e\}$

$\text{First}(A''') = \{d, e, \epsilon\}$

$\text{First}(B) = \{b, f\}$

$\text{First}(C) = \{c\}$



**FOLLOW(Z)={#}**

**FOLLOW(A)={#}**

**FOLLOW(A'')={d,e#}**

**FOLLOW(A')={d,e,#}**

**FOLLOW(A''')={#}**

**FOLLOW(B)={c,d,e,#}**

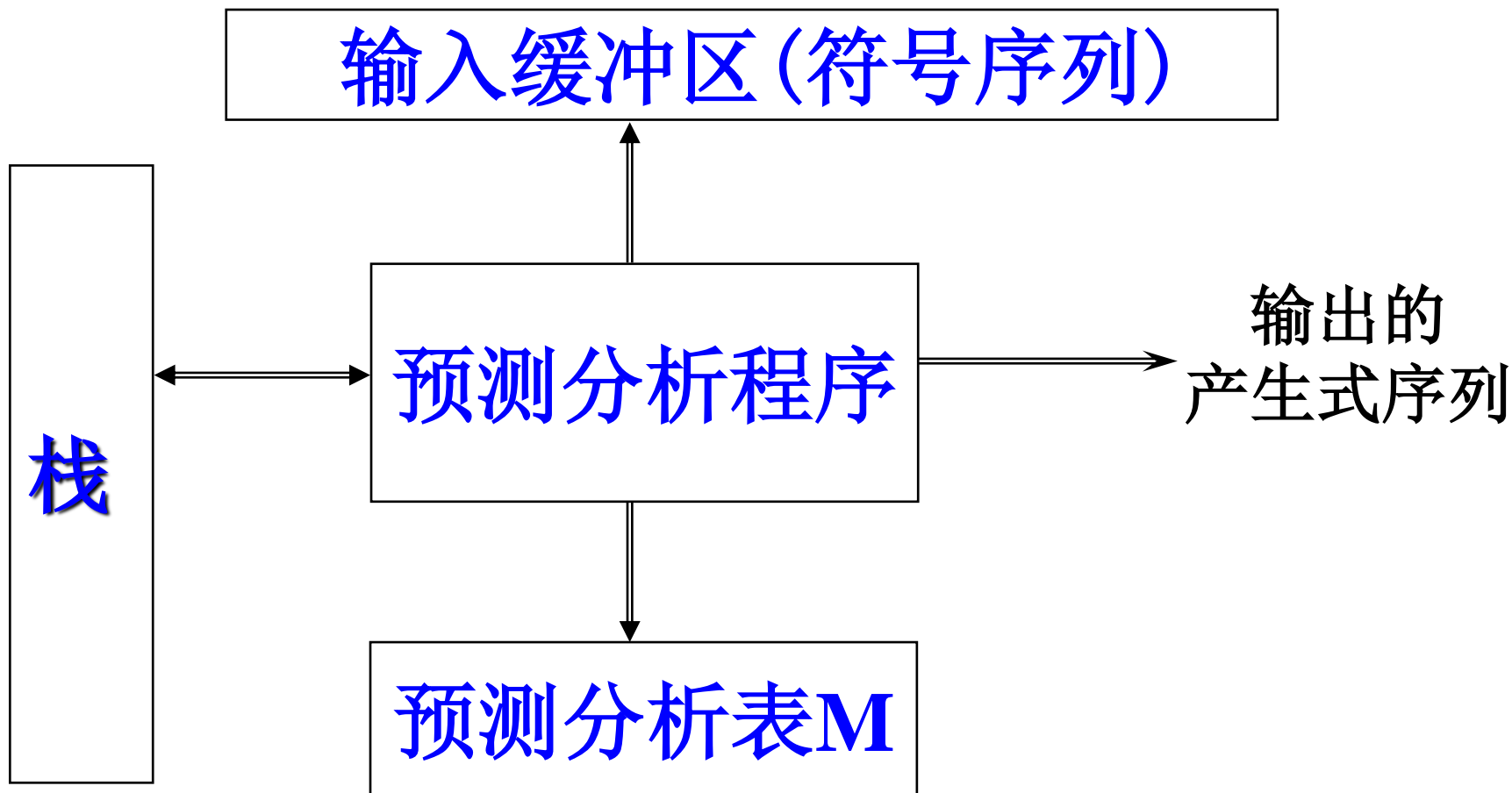
**FOLLOW(C)={c,d,e,#}**



## 4.3 预测分析法

- 系统维持一个分析表和一个分析栈，根据当前扫描到的符号，选择当前语法变量（处于栈顶）的候选式进行推导——希望找到相应输入符号串的最左推导。
- 一个通用的控制算法
- 一个分析栈，#为栈底符号
- 一个输入缓冲区，#为输入串结束符
- 一个统一形式的分析表M
  - 不同语言使用内容不同的分析表

## 4.3.1 预测分析器的构成





# 系统的执行与特点

- 在系统启动时，输入指针指向输入串的第一个字符，分析栈中存放着栈底符号#和文法的开始符号。
- 根据栈顶符号A和读入的符号a，查看分析表M, 以决定相应的动作。
- 优点：
  - 1) 效率高
  - 2) 便于维护、自动生成
- 关键——分析表M的构造



# 预测分析程序的总控程序

算法4.5 预测分析程序的总控程序。

输入：输入串 $w$ 和文法 $G=(V, T, P, S)$ 的分析表 $M$ ;

输出：如果 $w$ 属于 $L(G)$ ，则输出 $w$ 的最左推导，否则报告错误;

步骤：

1. 将栈底符号 $\#$ 和文法开始符号 $S$ 压入栈中;

2. repeat

3.        $X$ :=当前栈顶符号;

4.        $a$ :=当前输入符号;

5.       if  $X \in T \cup \{\#\}$  then

6.             if  $X=a$  then

7.                 {if  $X \neq \#$  then begin

8.                     将 $X$ 弹出栈;

9.                     前移输入指针

10.                    end}



# 预测分析程序的总控程序

```
11.                else error
12.                else
13.                if  $M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$  then begin
14.                将 $X$ 弹出栈;
15.                依次将 $Y_k, \dots, Y_2, Y_1$ 压入栈;
16.                输出产生式 $X \rightarrow Y_1 Y_2 \dots Y_k$ 
17.                end
18.                else error
19. until  $X = \#$ 
```

## 例4.10 考虑简单算术表达式文法的实现

$\text{FOLLOW}(E') = \{ ), \# \}$

$\text{FOLLOW}(T') = \{ +, ), \# \}$

$\text{FIRST}(TE') = \{ (, \text{id} \}$

$\text{FIRST}(+TE') = \{ + \}$

$\text{FIRST}(FT') = \{ (, \text{id} \}$

$\text{FIRST}(*FT') = \{ * \}$

$\text{FIRST}((E)) = \{ ( \}$

$\text{FIRST}(\text{id}) = \{ \text{id} \}$

$E \rightarrow TE' \quad E' \rightarrow +TE' | \epsilon \quad T \rightarrow FT'$

$T' \rightarrow *FT' | \epsilon \quad F \rightarrow (E) | \text{id}$

# 预测分析表

非终结符	输入符号					
	id	+	*	(	)	\$
<b>E</b>	$E \rightarrow TE'$			$E \rightarrow TE'$		
<b>E'</b>		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
<b>T</b>	$T \rightarrow FT'$			$T \rightarrow FT'$		
<b>T'</b>		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
<b>F</b>	$F \rightarrow id$			$F \rightarrow (E)$		



# 对输入串 $id+id*id$ 进行分析的过程

(在黑板上同时画出语法树)

栈	输入缓冲区	输出
#E	$id+id*id\#$	
#E' T	$id+id*id\#$	$E \rightarrow TE'$
#E' T' F	$id+id*id\#$	$T \rightarrow FT'$
#E' T' id	$id+id*id\#$	$F \rightarrow id$
#E' T'	$+id*id\#$	
#E'	$+id*id\#$	$T' \rightarrow \epsilon$
#E' T+	$+id*id\#$	$E' \rightarrow +TE'$
#E' T	$id*id\#$	





#E' T

id\*id#

#E' T' F

id\*id#

$T \rightarrow FT'$

#E' T' id

id\*id#

$F \rightarrow id$

#E' T'

\*id#

#E' T' F\*

\*id#

$T' \rightarrow *FT'$

#E' T' F

id#

#E' T' id

id#

$F \rightarrow id$

#E' T'

#

#E'

#

$T' \rightarrow \epsilon$

#

#

$E' \rightarrow \epsilon$

输出的产生式序列形成了最左推导对应的分析树

## 4.3.2 预测分析表的构造算法

算法4.6 预测分析表( $LL(1)$ 分析表)的构造算法。

输入：文法 $G$ ;

输出：分析表 $M$ ;

步骤:

1. 对 $G$ 中的任意一个产生式 $A \rightarrow \alpha$ , 执行第2步和第3步;
2. for  $\forall a \in \text{FIRST}(\alpha)$ , 将 $A \rightarrow \alpha$ 填入 $M[A, a]$ ;
3. if  $\varepsilon \in \text{FIRST}(\alpha)$  then  $\forall a \in \text{FOLLOW}(A)$ , 将 $A \rightarrow \alpha$ 填入 $M[A, a]$ ;  
if  $\varepsilon \in \text{FIRST}(\alpha) \& \# \in \text{FOLLOW}(A)$  then 将 $A \rightarrow \alpha$ 填入 $M[A, \#]$ ;
4. 将所有无定义的 $M[A, b]$ 标上出错标志。

# 预测分析表

非终结符	输入符号					
	id	+	*	(	)	\$
E	FIRST(T) = {(,id}					
E'	FIRST(F)= {(,id}					
T	T→ε }					
T'	T'→ε		T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		





# 预测分析法的实现步骤

---

1. 构造文法
2. 改造文法：消除二义性、消除左递归、提取左因子
3. 求每个候选式的**FIRST**集和变量的**FOLLOW**集
4. 检查是不是 **LL(1)** 文法  
若不是 **LL(1)**,说明文法的复杂性超过自顶向下方法的分析能力, 需要附加新的“信息”
5. 构造预测分析表
6. 实现预测分析器



### 4.3.3 预测分析的错误恢复

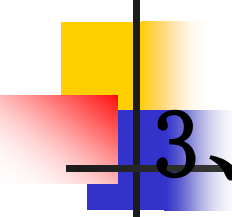
#### 1、发现错误

① 栈顶的终结符与当前输入符不匹配

② 非终结符A位于栈顶，面临的输入符为a，但分析表M的 $M[A, a]$ 为空

#### 2、“应急”恢复策略

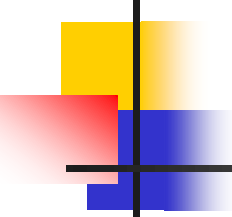
跳过输入串中的一些符号直至遇到“同步符号”为止。



### 3、同步符号的选择

①把FOLLOW(A)中的所有符号作为A的同步符号。跳过输入串中的一些符号直至遇到这些“同步符号”，把A从栈中弹出，可使分析继续

②把FIRST(A)中的符号加到A的同步符号集，当FIRST(A)中的符号在输入中出现时，可根据A恢复分析



③可以把表示语句开始的一些关键字加入到同步记号集中

④如果栈顶的终结符不能被匹配，就可以弹出该终结符，此时相当于把所有的符号都看作同步符号

用synch 表示由相应非终结符的FOLLOW集得到的同步符号，则前面的预测分析表变为：



$$\begin{aligned} \text{FOLLOW}(F) &= \text{FIRST}(E') \cup \text{FIRST}(T') \\ &= \{ +, ), *, \epsilon \} \end{aligned}$$

$$\begin{aligned} \text{FOLLOW}(T') &= \text{FOLLOW}(T) \\ &= \text{FIRST}(E') \cup \text{FOLLOW}(E) \\ &= \{ +, ), \# \} \end{aligned}$$

$$\begin{aligned} \text{FOLLOW}(T) &= \text{FIRST}(E') \cup \text{FOLLOW}(E) \\ &= \{ +, ), \# \} \end{aligned}$$

$$\text{FOLLOW}(E') = \text{FOLLOW}(E) = \{ ), \# \}$$

# 不含错误处理的分析表

非终结符	输入符号					
	id	+	*	(	)	#
<b>E</b>	$E \rightarrow TE'$			$E \rightarrow TE'$		
<b>E'</b>		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
<b>T</b>	$T \rightarrow FT'$			$T \rightarrow FT'$		
<b>T'</b>		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
<b>F</b>	$F \rightarrow id$			$F \rightarrow (E)$		

# 加入错误处理的分析表

非终结符	输入符号					
	id	+	*	(	)	#
<b>E</b>	$E \rightarrow TE'$			$E \rightarrow TE'$	<b>synch</b>	<b>synch</b>
<b>E'</b>		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
<b>T</b>	$T \rightarrow FT'$	<b>synch</b>		$T \rightarrow FT'$	<b>synch</b>	<b>synch</b>
<b>T'</b>		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
<b>F</b>	$F \rightarrow id$	<b>synch</b>	<b>synch</b>	$F \rightarrow (E)$	<b>synch</b>	<b>synch</b>

# 句子)id+\*id的分析过程

栈	输入	备注
#E	)id*+id#	错误, 跳过) id在FIRST(E)中
#E	id*+id#	
#E'T	id*+id#	
#E'T'F	id*+id#	
#E'T'id	id*+id#	
#E'T'	*+id#	错误,M[F,+]=synch F 已被弹出
#E'T'F*	*+id#	
#E'T'F	+id#	
#E'T'	+id#	
#E'	+id#	
#E'T+	+id#	
#E'T	id#	
#E'T'F	id#	
#E'T'id	id#	
#E'T'	#	
#E'	#	
#	#	



## 4.4 递归下降分析法— 一个设想

1. 为每个非终结符，编写一个可以递归调用的处理子程序，名字就是该非终结符

$$A \rightarrow X_1 X_2 \dots X_k \dots X_n$$

2. 程序体按产生式的右端来编写

- (1) 当遇到 $X_k$ 是终极符号时直接进行匹配;
- (2) 当遇到 $X_k$ 是语法变量时就调用 $X$ 对应的处理子程序.

## 4.4.1 递归下降分析法的基本思想

例4.14 对于产生式 $E' \rightarrow +TE'$ ，与 $E'$ 对应的子程序可以按如下方式来编写：

**procedure**  $E'$

**begin**

*match*('+');

$T$ ;

$E'$

**end**;

*/\*调用识别 $T$ 的过程\*/*

*/\*调用识别 $E'$ 的过程\*/*

$E \rightarrow TE'$

$E' \rightarrow +TE' | \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' | \varepsilon$

$F \rightarrow (E) | id$



## 4.4.1 递归下降分析法的基本思想

---

其中，服务子程序 $match$ 用来匹配当前的输入记号，其代码为：

```
procedure match(t:token);  
begin  
    if lookhead=t then  
        lookhead:=nexttoken;  
    else error          /*调用出错处理程序*/  
end;
```



## 4.4.2 语法图和递归子程序法

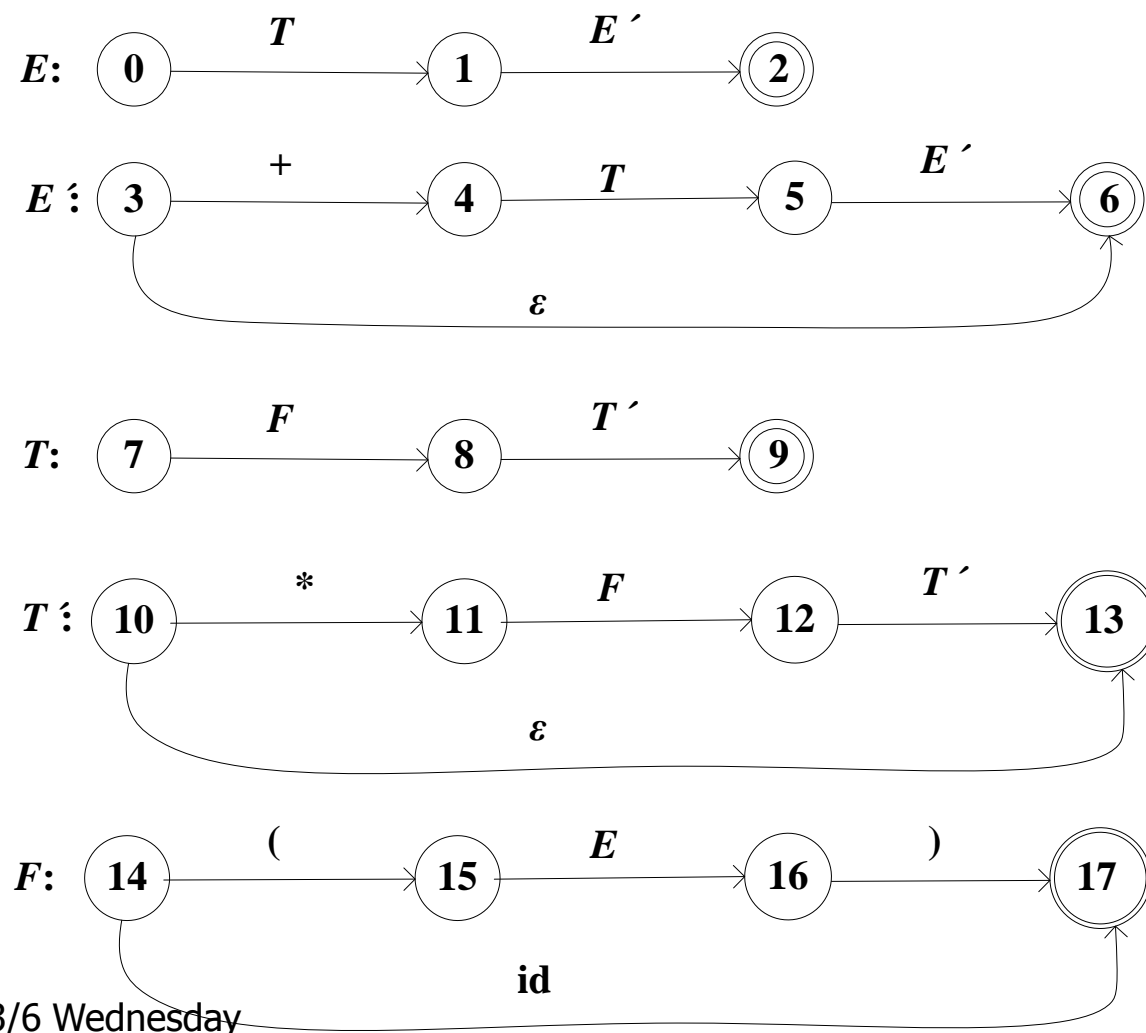
- 状态转换图（语法图）是非常有用的设计工具
- 语法分析器和词法分析器的状态转换图不同
  - 每个非终结符对应一个状态转换图，边上的标记是记号和非终结符
  - 记号上的转换意味着如果该记号是下一个输入符号，就应进行转换
  - 非终结符A上的转换是对与A对应的过程的调用



## 4.4.2 语法图和递归子程序法

- 从文法构造语法图，对每个非终结符A执行如下操作
  - 创建一个开始状态和一个终止状态（返回状态）
  - 对每个产生式 $A \rightarrow X_1 X_2 \dots X_n$ ，创建一条从开始状态到终止状态的路径，边上的标记分别为 $X_1$ ,  $X_2, \dots, X_n$

## 例4.15 简单表达式文法的语法图



$E \rightarrow TE'$   
 $E' \rightarrow +TE' | \epsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' | \epsilon$   
 $F \rightarrow (E) | id$

## 4.4.3 基于语法图的语法分析器工作方式

- 初始时，分析器进入状态图的开始状态，输入指针指向输入符号串的第一个符号。
- 如果经过一些动作后，它进入状态 $s$ ，且从状态 $s$ 到状态 $t$ 的边上标记了终结符 $a$ ，此时下一个输入符又正好是 $a$ ，则分析器将输入指针向右移动一位，并进入状态 $t$ 。

### 4.4.3 基于语法图的语法分析器工作方式

- 另一方面，如果边上标记的是非终结符A，则分析器进入A的初始状态，但不移动输入指针。一旦到达A的终态，则立刻进入状态t，事实上，分析器从状态s转移到状态t时，它已经从输入符号串“读”了A（调用A对应的过程）。
- 最后，如果从s到t有一条标记为  $\epsilon$  的边，那么分析器从状态s直接进入状态t而不移动输入指针。



## 4.4.4 语法图的化简与实现

### (1) 左因子提取

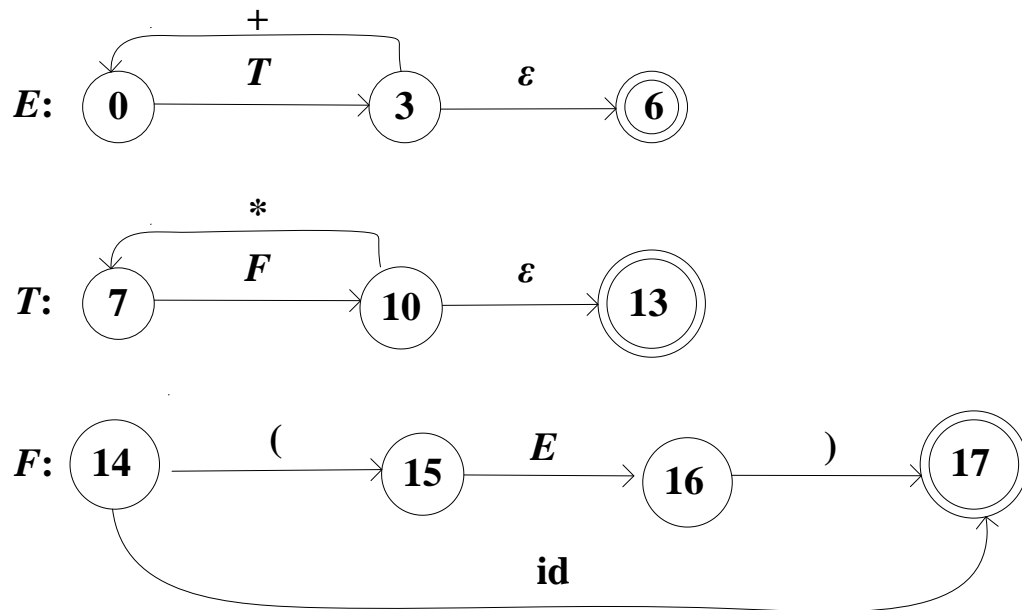
将形如 $A \rightarrow YX|YZ$ 的产生式替换为 $A \rightarrow Y(X|Z)$ ;

### (2) 右因子提取

将形如 $A \rightarrow YX|ZX$ 的产生式替换为 $A \rightarrow (Y|Z)X$ ;

### (3) 尾递归消除

将形如 $X \rightarrow YX|Z$ 的产生式替换为 $X \rightarrow Y^*Z$ 。





## 例4.16 简单算术表达式的语法分析器

- E 的子程序( $E \rightarrow T(+T)^*$ )

**procedure E;**

**begin**

**T;**                      T 的过程调用

**while lookahead='+' do**

**begin**                      当前符号等于+时

**match('+');**              处理终结符+

**T**                      T 的过程调用

**end**

**end;**                      lookahead: 当前符号



# T 的子程序 ( $T \rightarrow F (*F) *$ )

---

**procedure T;**

**begin**

**F;**                      F 的过程调用

**while lookahead='\*' then**

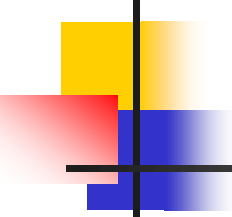
**begin**                      当前符号等于 \* 时

**match('\*');**              处理终结符 \*

**F**                      F 的递归调用

**end**

**end;**



# F 的子程序( $F \rightarrow (E)|id$ )

```
procedure F;  
begin  
  if lookahead='(' then  
    begin          当前符号等于 (  
      match('(');  处理终结符 (  
      E;           E 的递归调用  
      match(')');  处理终结符)  
    end  
  else if lookahead=id then  
    match(id)      处理终结符id  
  else error      出错处理  
end
```



# 主程序

---

begin

lookhead:=nexttoken; 调词法分析程序  
E E 的过程调用

end

## 服务子程序

**procedure match(t:token);**

**begin**

**if lookhead=t then**

**lookhead:=nexttoken**

**else error** 出错处理程序

**end;**



## 4.4.5 递归子程序法的实现步骤

- 1) 构造文法;
- 2) 改造文法: 消除二义性、消除左递归、提取左因子;
- 3) 求每个候选式的**FIRST**集和语法变量的**FOLLOW**集;
- 4) 检查 $G$ 是不是  $LL(1)$  文法, 若 $G$ 不是  $LL(1)$ 文法, 说明文法 $G$ 的复杂性超过了自顶向下方法的分析能力, 需要附加新的“信息”;
- 5) 按照 $LL(1)$ 文法画语法图;
- 6) 化简语法图;
- 7) 按照语法图为每个语法变量设置一个子程序。



# 递归子程序法的优缺点分析

- 优点：
  - 1) 直观、简单、可读性好
  - 2) 便于扩充
- 缺点：
  - 1) 递归算法的实现效率低
  - 2) 处理能力相对有限
  - 3) 通用性差，难以自动生成
- 从递归子程序法及FIRST与FOLLOW集看如何进一步用好当前的输入符号？



# 本章小结

---

1. 自顶向下分析法和自底向上分析法分别寻找输入串的最左推导和最左归约
2. 自顶向下分析会遇到二义性问题、回溯问题、左递归引起的无穷推导问题，需对文法进行改造：消除二义性、消除左递归、提取公共左因子
3.  $LL(1)$ 文法是一类可以进行确定分析的文法，利用FIRST集和FOLLOW集可以判定某个上下文无关文法是否为 $LL(1)$ 文法





# 本章小结

---

4.  $LL(1)$ 文法可以用 $LL(1)$ 分析法进行分析。
5. 递归下降分析法根据各个候选式的结构为每个非终结符编写一个子程序。
6. 使用语法图可以方便地进行递归子程序的设计。

文法G:

$A \rightarrow [B$

$B \rightarrow X] | BA$

$X \rightarrow Xa | Xb | a | b$

构造其

LL (1) 分析表

消除左递归

**FIRST**

$A \rightarrow [B$

$\{[$

$B \rightarrow X]B'$

$\{a, b\}$

$B' \rightarrow AB'$

$\{[$

$B' \rightarrow \epsilon$

$\{\epsilon\}$

$X \rightarrow aX'$

$\{a\}$

$X \rightarrow bX'$

$\{b\}$

$X' \rightarrow aX'$

$\{a\}$

$X' \rightarrow bX'$

$\{b\}$

$X' \rightarrow \epsilon$

$\{\epsilon\}$



---

**FIRST(A)={[]}**

**FIRST(B)={a,b}**

**FIRST(B')={[,  $\epsilon$ ]**

**FIRST(X)={a,b}**

**FIRST(B)={a,b,  $\epsilon$ }**



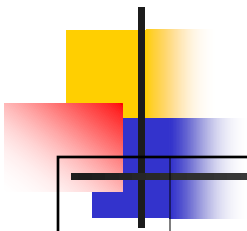
**Follow(A)={[, \$}**

**Follow(B)={[, \$}**

**Follow(B')={[, \$}**

**Follow(X)={]}**

**Follow(X')={]}**



	a		b	[	]	\$
A				$A \rightarrow [B$		
B	$B \rightarrow X B'$	$B \rightarrow X B'$				
B'				$B' \rightarrow AB'$ $B' \rightarrow \epsilon$		$B' \rightarrow \epsilon$
X	$X \rightarrow aX'$	$X \rightarrow bX'$				
X'	$X' \rightarrow aX'$	$X' \rightarrow bX'$			$X' \rightarrow \epsilon$	

# 预测分析表

非终结符	输入符号						
	a	b	c	d	e	f	\$
Z	1						
A	2						
A''		3	4			3	
A'				5	6		7
B		8				9	
C			10				