



# 第四章 语法分析—— 自上而下分析



## 内容线索

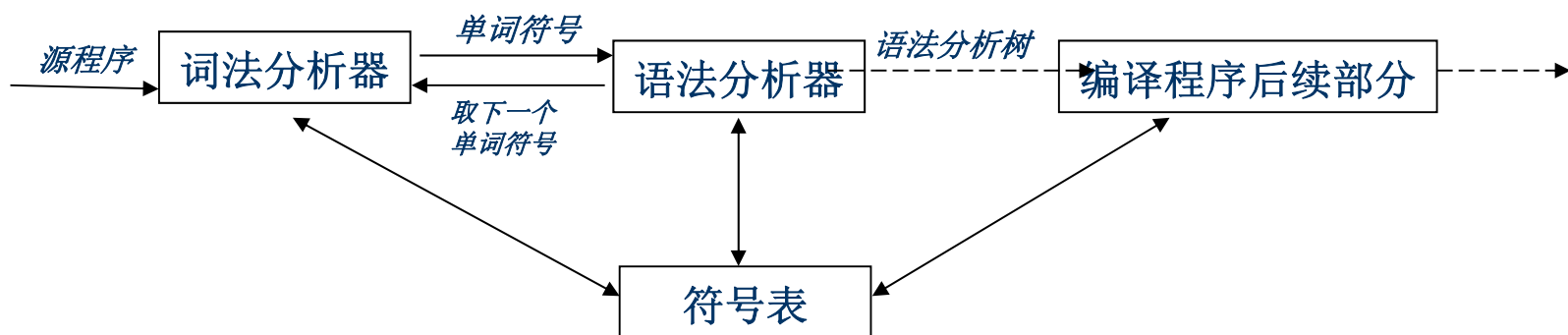
- 语法分析器的功能
- 自上而下分析方法概述
- LL(1) 分析方法
- 递归下降分析程序
- 预测分析程序

# 语法分析器

- 语法分析的任务：

对任一给定  $w \in V_T^*$ ，判断  $w \in L(G)$ ?

- 语法分析器：按照产生式规则，做识别  $w$  的工作



语法分析器在编译程序中的地位



# 语法分析方法

## ■ 自上而下分析

- LL(1) 分析法
- 递归下降分析法
- 预测分析法

从文法的开始符号出发，反复使用各种产生式，寻找与输入符号匹配的最左推导。

## ■ 自下而上分析

- 算符优先分析法
- LR 分析法

从输入符号串开始，逐步进行归约（最右推导的逆过程），直至归约到文法的开始符号。



## 内容线索

- ✓ 语法分析器的功能
- 自上而下分析方法概述
- LL(1) 分析方法
- 递归下降分析程序
- 预测分析程序



## 自上而下分析

- 从文法的开始符号出发，向下推导，推出句子
- 对任何的输入串(单词符号)，试图用一切可能的办法，从文法的开始符号出发，自上而下地为输入串建立一棵语法树，即为输入串寻找一个最左推导。

## 语法分析示例

**$G[S]: S \rightarrow id:=E \quad E \rightarrow E+E \quad E \rightarrow E * E$**   
 **$E \rightarrow -E \quad E \rightarrow (E) \quad E \rightarrow id$**

**$id:=-id$  是否为  $G[S]$  的句子?**

**$S \Rightarrow id:=E \Rightarrow id:= -E \Rightarrow id:= -id$**

基本思想:

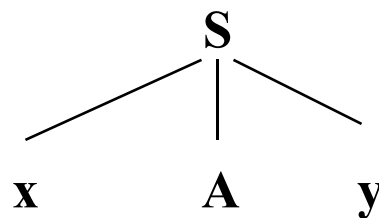
最左推导过程中, 为当前字符选择一个产生式, 根据产生式右部首字符进行匹配

例. 设文法  $G[S]: S \rightarrow xAy, A \rightarrow ** \mid *$

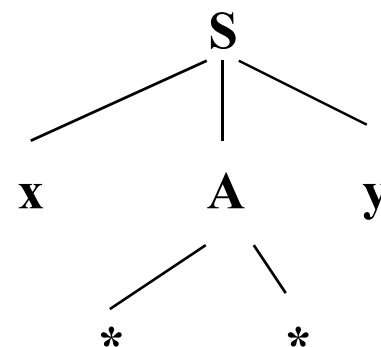
判定输入串  $x * y$  是否为它的句子?

$x * y$   
↑

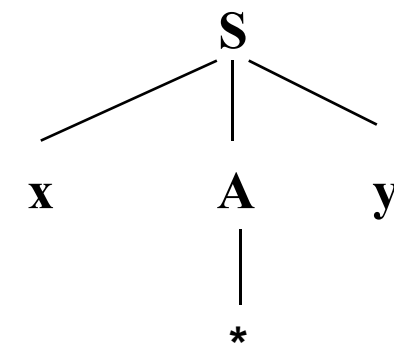
用  $S \rightarrow xAy$



用  $A \rightarrow **$   
(回溯)



用  $A \rightarrow *$   
(成功)



推导过程:

$S \Rightarrow xAy$

$\Rightarrow x**y$  (回溯)

$\Rightarrow x*y$  (成功)



## 语法分析示例

**$G[S]: S \rightarrow id:=E \quad E \rightarrow E+E \quad E \rightarrow E * E$**

**$E \rightarrow -E \quad E \rightarrow (E) \quad E \rightarrow id$**

**$id:=-id+id+id$  是否为  $G[S]$  的句子?**

**$S \Rightarrow id:=E \Rightarrow id:= -E \Rightarrow id:= - id$  ( 错误 )**

**$S \Rightarrow id:=E \Rightarrow id:= -E \Rightarrow id:= - E+E$**

**$\Rightarrow id:= - E+E+E \Rightarrow id:= - E+ E+E+E \Rightarrow \dots$**

# 带回溯自上而下分析面临的问题

- 虚假匹配问题

- 回溯

- 回溯会引起时间和空间的大量消耗

- 文法的左递归问题

- 一个文法是含有左递归的，如果存在非终结符P

$$P \Rightarrow^+ P\alpha$$

- 含有左递归的文法将使自上而下的分析过程陷入无限循环

- 报告分析不成功时，难于知道输入串中出错的确切位置。

实际上采用了一种穷尽一切可能的试探法，因此效率很低，代价很高



## 内容线索

- 语法分析器的功能
- 自上而下分析方法概述
- ✓ **LL(1) 分析方法**
- 递归下降分析程序
- 预测分析程序
- LL(1) 分析中的错误处理



# LL(1)分析法

- 从左(**L**eft)到右扫描输入串；构造最左(**L**eftmost)推导；分析时每步向前看一个(**1**)字符。
- 目的：构造不带回溯的自上而下分析算法
  - 左递归的消除
  - 消除回溯，提左因子
  - **FIRST**集合，**FOLLOW**集合
  - **LL(1)**分析条件
  - **LL(1)**分析方法

# 左递归文法

- 一个文法含有下列形式的产生式时，

a) 直接递归

$$A \rightarrow A\beta \quad A \in V_N, \beta \in V^*$$

b) 间接递归

$$A \rightarrow B\beta$$

$$B \rightarrow A\alpha \quad A, B \in V_N, \alpha, \beta \in V^*$$

称为左递归文法。

- 如果一个文法是左递归时，则不能采用自顶向下分析法。



例1. 文法  $S \rightarrow Sa$

$S \rightarrow b$

是直接左递归

语言是:  $L = \{ ba^n \mid n \geq 0 \}$

例2. 文法  $P_1 \rightarrow aP_2$

$P_1 \rightarrow P_2b$

$P_2 \rightarrow P_1c$

$P_2 \rightarrow d$

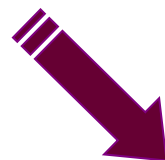
是间接左递归

# 消除直接左递归

$P \rightarrow P\alpha \mid \beta$  ( $\alpha \neq \epsilon$ ,  $\beta$  不以  $P$  开头)



$P \rightarrow \beta P'$   
 $P' \rightarrow \alpha P' \mid \epsilon$



$\beta$

$\beta\alpha$

$\beta\alpha\alpha$

$\beta\alpha\alpha\alpha$

... ..



例. 文法  $E \rightarrow E+T \mid T$

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

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



$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

- 一般地, 假定关于  $P$  的产生式是

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

其中:  $\alpha_i \neq \varepsilon$ ,  $\beta_i$  不以  $P$  开头,

则改写为:  $P \rightarrow \beta_1 P' \mid \beta_2 P' \mid \dots \mid \beta_n P'$

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

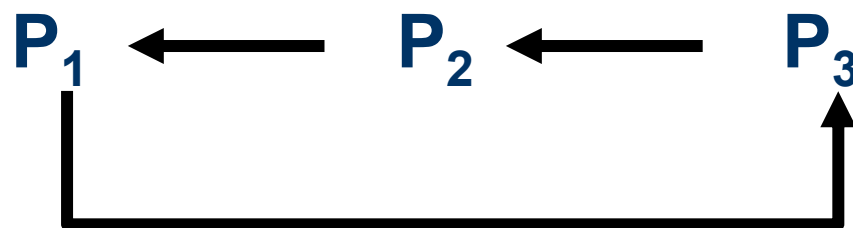


## 消除左递归的算法思想

文法  $P_2 \rightarrow P_1 b$

$P_3 \rightarrow P_2 c$

$P_1 \rightarrow P_3 d$



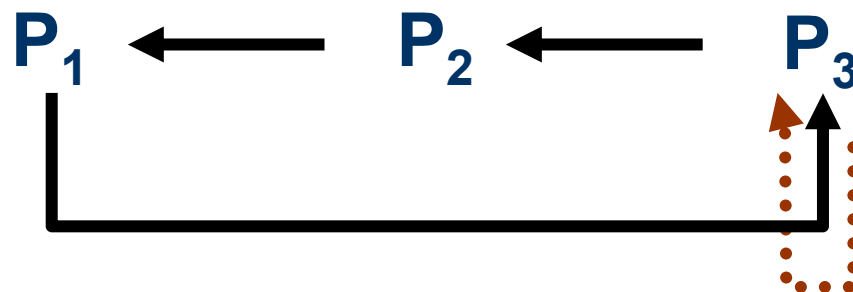
文法  $P_1 \rightarrow P_3 d$

$P_2 \rightarrow P_1 b$

$P_3 \rightarrow P_2 c$

$\Rightarrow P_1 bc$

$\Rightarrow P_3 dbc$



# 消除左递归算法

(1) 排序:  $P_1, P_2, \dots, P_n$

(2) **FOR**  $i := 1$  **TO**  $n$  **DO**

**BEGIN**

**FOR**  $j := 1$  **TO**  $i - 1$  **DO**

把形如  $P_i \rightarrow P_j Y$  的规则改写为:

$P_i \rightarrow \delta_1 Y \mid \delta_2 Y \mid \dots \mid \delta_k Y$

其中:  $P_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$  是关于  $P_j$   
的所有规则;

消除关于  $P_i$  规则的直接左递归。

**END**

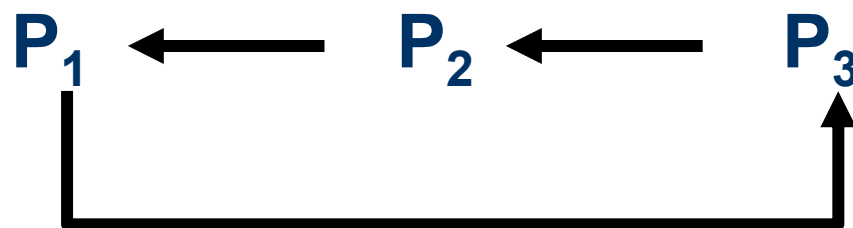
(3) 化简: 删除永不使用的产生式

## 算法示意

文法  $P_2 \rightarrow P_1 b$

$P_3 \rightarrow P_2 c$

$P_1 \rightarrow P_3 d$



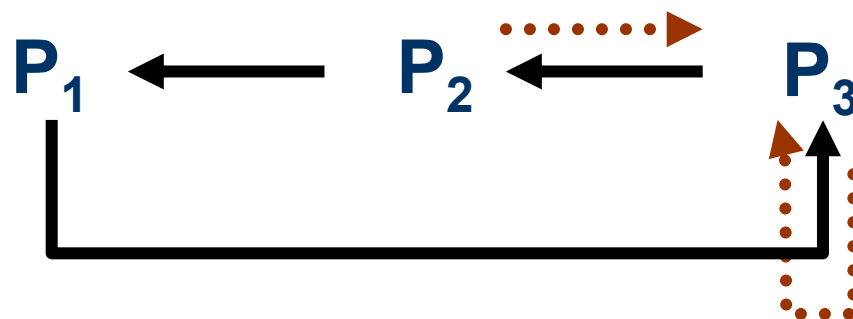
文法  $P_1 \rightarrow P_3 d$

$P_2 \rightarrow P_1 b$

$\Rightarrow P_3 db$

$P_3 \rightarrow P_2 c$

$\Rightarrow P_3 dbc$



例.文法  $G[P_3]$ :  $P_3 \rightarrow P_2c|c$     $P_2 \rightarrow P_1b|b$     $P_1 \rightarrow P_3a|a$

有推导:  $P_3 \Rightarrow P_2c \Rightarrow P_1bc \Rightarrow P_3abc$ , 存在左递归。

按  $P_1$ (1)、 $P_2$ (2)、 $P_3$ (3) 排序, 执行算法得:

$i=1$ ,  $j$  从 1 至 0,  $P_1$  的产生式  $P_1 \rightarrow P_3a|a$  无直接左递归, 无需消除直接左递归。

$i=2$ ,  $j$  从 1 至 1,  $P_2$  的候选式含  $P_1$ ,  $P_1$  的产生式代入  $P_2$  的产生式得:

$P_2 \rightarrow P_3ab|ab|b$ , 无直接左递归。

$i=3$ ,  $j$  从 1 至 2:

$j=1$ ,  $P_3$  的候选式不含  $P_1$ , 所以无需替换;

$j=2$ ,  $P_3$  的候选式含  $P_2$ , 将  $P_2 \rightarrow P_3ab|ab|b$  代入  $S$  的候选式得:

$$P_3 \rightarrow P_3abc|abc|bc|c$$

再消除直接左递归得:

$$P_3 \rightarrow abcP_3'|bcP_3'|cP_3'$$
$$P_3' \rightarrow abcP_3'|\epsilon$$

消除无用产生式:  $P_2 \rightarrow P_3ab|ab|b$ ,  $P_1 \rightarrow P_3a|a$ ,

得文法:  $P_3 \rightarrow abcP_3'|bcP_3'|cP_3'$

$$P_3' \rightarrow abcP_3'|\epsilon$$

文法对应的正规式:  $V1=(abc|bc|c)(abc)^*$ 。

例. 文法  $G[S]$ :  $S \rightarrow Qc|c$      $Q \rightarrow Rb|b$      $R \rightarrow Sa|a$

解: 1) 排序:  $S(1)$ 、 $Q(2)$ 、 $R(3)$

2) 代入得:  $S \rightarrow Qc|c$

$Q \rightarrow Rb|b$

$R \rightarrow Rbca|bca|ca|a$

消除直接左递归:

$S \rightarrow Qc|c$

$Q \rightarrow Rb|b$

$R \rightarrow bcaR' | caR' | aR'$

$R' \rightarrow bcaR' | \epsilon$

消除隐含的左递归算法与非终极符排序方法无关



## 回溯问题

例如，有产生式：

语句  $\rightarrow$  **if** 条件 **then** 语句 **else** 语句  
          | **while** 条件 **do** 语句  
          | **begin** 语句表 **end**

若要寻找一个语句，那么关键字 **if**, **while**,  
**begin** 就提示某个替换式是唯一的替换式。

**无回溯！**



## 回溯问题

例如，有产生式：

语句  $\rightarrow$  **if** 条件 **then** 语句 **else** 语句

| **if** 条件 **then** 语句

| **while** 条件 **do** 语句

| **begin** 语句表 **end**

产生回溯！



## 回溯原因

若当前符号 =  $\mathbf{a}$ ，下一步要展开  $\mathbf{A}$ ，

而  $\mathbf{A} \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ ，

怎样选择  $\alpha_i$ ？

(1) 以  $\mathbf{a}$  为头的  $\alpha_i$  如果只有一个，则替换唯一；

(2) 若以  $\mathbf{a}$  为头有多个  $\alpha_i$  的，则替换不唯一，可能需要回溯，这是文法的问题，应该变换文法。



# 文法的要求

(1) 不含左递归

(2) 对每个非终结符的候选式，其任何推导的头符号（终结符）集合两两不相交。

- 符号串 $\alpha$ 的终结首符集**FIRST**( $\alpha$ ) 定义为：

$$\text{FIRST}(\alpha) = \{ a \mid \alpha \xRightarrow{*} a \dots, a \in V_T \}$$

特别地，若 $\alpha \xRightarrow{*} \varepsilon$ ，则规定 $\varepsilon \in \text{FIRST}(\alpha)$ 。

- 以上条件（2）可表示为：对文法的任一非终结符号 $A$ ，若

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

则应有 **FIRST**( $\alpha_i$ )  $\cap$  **FIRST**( $\alpha_j$ ) =  $\Phi$ ,  $i \neq j$

## 回溯解决方法

- 提取公共左因子，将文法改造成任何非终结符的所有候选首符集两两不相交。

$$A \rightarrow \delta\beta_1 \mid \delta\beta_2 \mid \dots \mid \delta\beta_n \mid \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_m$$

(其中 $\gamma_1$ 、 $\gamma_2$ 、 $\dots$ 、 $\gamma_m$ 不以 $\delta$ 开头)



$$A \rightarrow \delta A' \mid \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_m$$
$$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$



例. 文法 **G**:  $S \rightarrow aSb | aS | \epsilon$

解: 提取:  $S \rightarrow aS( b | \epsilon)$

$$S \rightarrow \epsilon$$

引入新符:  $S \rightarrow aSA | \epsilon$

$$A \rightarrow b | \epsilon$$

一般地, 经过反复提取左因子可把每一个非终结符的所有候选首符集变成两两不相交。

# LL (1) 分析条件

- 若文法已经消除了左递归，且对每个非终结符满足  $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \Phi$ 
  - 是否就解决了 虚假匹配 和 回溯 的问题？
- 对某个输入符号  $a$ ，及待匹配的非终结符  $A$   
(  $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$  )
  - $a$  属于某个候选式的 **FIRST** 集合
  - $a$  不属于任何候选式的 **FIRST** 集合，即对任意  $\alpha_i$ ， $a \notin \text{FIRST}(\alpha_i)$

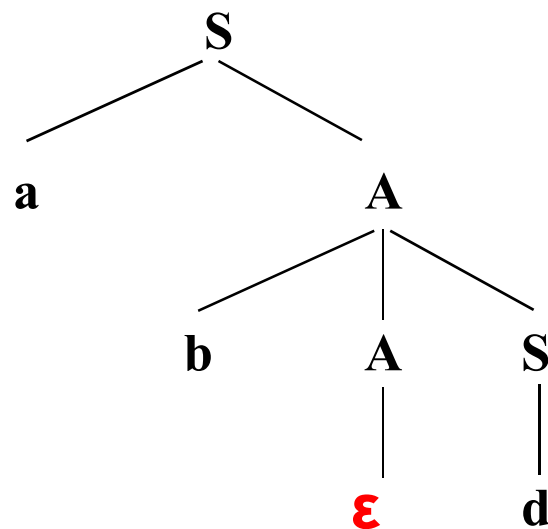
## 示例

$\text{FIRST}(S) = \{a, d\}$

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

例.  $G(S): S \rightarrow aA|d$       输入符号串 **abd** 是否为句子?  
 $A \rightarrow bAS|d|\epsilon$

$S \Rightarrow aA$	$S \Rightarrow aA$
$\Rightarrow abAS$	$\Rightarrow abAS$
$\Rightarrow abdS$	$\Rightarrow abS$
。 。 。	$\Rightarrow abd$



这是因为 **A** 有产生式  $A \rightarrow \epsilon$ , 而从开始符号 **S** 可以得出

$S \xRightarrow{*} \dots Ad \dots$

# FOLLOW

- 设**S**是文法**G**的开始符号，对**G**的任何非终结符**A**，定义**A**的后继终结符号集为：

$$\text{FOLLOW}(\mathbf{A}) = \{ \mathbf{a} \mid \mathbf{S} \xRightarrow{*} \dots \mathbf{Aa} \dots, \mathbf{a} \in \mathbf{V}_T \}$$

- 特别地,若 $\mathbf{S} \xRightarrow{*} \dots \mathbf{A}$ ，则规定

$$\# \in \text{FOLLOW}(\mathbf{A}).$$

**FOLLOW(A)**是所有句型中出现在紧接**A**之后的终结符或“#”。

## 结论

- 当非终结符**A**面临输入符号**a**，且 **$a \notin \text{FIRST}(\alpha_i)$**  (对任意**i**) 时，如果**A**的某个候选首符集包含 **$\epsilon$**  (即 **$\epsilon \in \text{FIRST}(\mathbf{A})$** )，那么，当 **$a \in \text{FOLLOW}(\mathbf{A})$**  时，就允许**A**自动匹配 (即选用 **$\mathbf{A} \rightarrow \epsilon$** 工作)，否则，认为**a**的出现是一种语法错误。
- 要正确进行不带回溯的语法分析，文法应满足的第三个条件可表示为：若**A**的候选首符集中包含 **$\epsilon$** ，则

$$\text{FIRST}(\mathbf{A}) \cap \text{FOLLOW}(\mathbf{A}) = \Phi$$

# LL (1) 文法

■ 如果文法**G**满足以下条件:

(1) 文法消除了左递归;

(2) 文法中每个非终结符**A**的各个产生式的候选首符集两两不相交, 即

若  $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ ,

则  $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \Phi$ , ( $i \neq j$ );

(3) 对文法中的每个非终结符**A**, 若它存在某个候选首符集中包含 $\epsilon$ , 则  $\text{FIRST}(A) \cap \text{FOLLOW}(A) = \Phi$ ,

则称该文法**G**为**LL(1)文法**。



# LL (1) 分析方法

- 对一个LL (1) 文法，可以对某个输入串进行有效的无回溯的自上而下分析。
- 设面临的输入符号为 $a$ ，要用非终结符 $A$ 进行匹配，且 $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ ，则可如下分析：
  - (1) 若 $a \in \text{FIRST}(\alpha_i)$ ，则指派 $\alpha_i$ 执行匹配任务；
  - (2) 否则
    - 1) 若 $\epsilon \in \text{FIRST}(A)$ ，且 $a \in \text{FOLLOW}(A)$ ，则让 $A$ 与 $\epsilon$ 自动匹配；
    - 2) 否则， $a$ 的出现是一种语法错误。

## 示例

**$G(S): S \rightarrow aA|d, A \rightarrow bAS|d|\epsilon$** 是否为LL(1)文法?

解: (1) 不含左递归;

(2) 对于S,  **$\text{First}(aA)=\{a\}$**   **$\text{First}(d)=\{d\}$**

对于A,  **$\text{First}(bAS)=\{b\}$**   **$\text{First}(d)=\{d\}$**   **$\text{First}(\epsilon)=\{\epsilon\}$**

任一非终结符的候选首符集两两不相交

满足条件 (2)

(3)  **$\text{FIRST}(S)=\{a, d\}$**   **$\text{FIRST}(A)=\{b, d, \epsilon\}$**

**$\text{Follow}(S)=\{\#, a, d\}$**   **$\text{Follow}(A)=\{a, d, \# \}$**

所以  **$\text{FIRST}(A) \cap \text{Follow}(A)=\{d\} \neq \Phi$**

不满足条件 (3)

所以, 该文法不是LL(1)文法

## 示例

如果文法为  $G(S): S \rightarrow aA|d, A \rightarrow bAS|\epsilon$  是否为 LL(1) 文法?

解: (1) 不含左递归;

(2) 对于  $S$ ,  $\text{First}(aA)=\{a\}$   $\text{First}(d)=\{d\}$

对于  $A$ ,  $\text{First}(bAS)=\{b\}$   $\text{First}(\epsilon)=\{\epsilon\}$

任一非终结符的候选首符集两两不相交

满足条件 (2)

(3)  $\text{FIRST}(S)=\{a, d\}$   $\text{FIRST}(A)=\{b, \epsilon\}$

$\text{Follow}(S)=\{\#, a, d\}$   $\text{Follow}(A)=\{\#, a, d\}$

满足条件 (3)

所以, 该文法是 LL(1) 文法

## 示例

修改文法为  $G(S): S \rightarrow aA|d, A \rightarrow bAS|\epsilon$

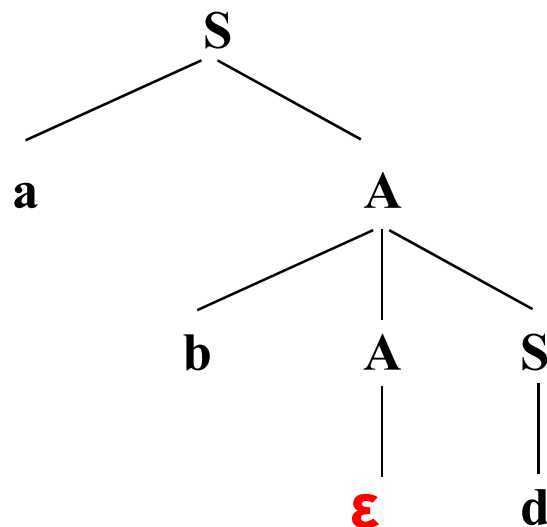
输入符号串 **abd** 是否为句子?

$S \Rightarrow aA$

$\Rightarrow abAS$

$\Rightarrow abS$

$\Rightarrow abd$



## FIRST ( $\alpha$ ) 构造

对于符号串  $\alpha = X_1X_2\cdots X_n$ , 构造 FIRST ( $\alpha$ )

- (1) 置  $\text{FIRST}(\alpha) = \text{FIRST}(X_1) - \{\epsilon\}$ ;
- (2) 若对  $1 \leq j \leq i-1$ ,  $\epsilon \in \text{FIRST}(X_j)$ , 则把  $\text{FIRST}(X_i) - \{\epsilon\}$  加到  $\text{FIRST}(\alpha)$  中;
- (3) 若对  $1 \leq j \leq n$ ,  $\epsilon \in \text{FIRST}(X_j)$ , 则把  $\epsilon$  加到  $\text{FIRST}(\alpha)$  中。

# FIRST (X) 构造

对于文法**G**的每个文法符号 $X \in V_T \cup V_N$ ，构造**FIRST (X)**的方法是：

(1) 若 $X \in V_T$ ，则**FIRST (X) = {X}**;

(2) 若 $X \in V_N$ ，且有产生式 $X \rightarrow a...$ ， $a \in V_T$ ，则把 $a$ 加入到**FIRST (X)**中，若有 $X \rightarrow \epsilon$ ，则把 $\epsilon$ 加入**FIRST (X)**;

(3) 若 $X \in V_N$ ，且 $X \rightarrow Y \dots$ ， $Y \in V_N$ ，则把


**FIRST (Y) - { $\epsilon$ }**加到**FIRST (X)**中，

若 $X \rightarrow Y_1 Y_2 \dots Y_k$ ， $Y_1, Y_2, \dots, Y_{i-1} \in V_N$ ， $\epsilon \in \text{FIRST}(Y_i)$ ，则把  
( $1 \leq j \leq i-1$ )

**FIRST (Y<sub>i</sub>) - { $\epsilon$ }**加到**FIRST (X)**中。

特别地，若 $\epsilon \in \text{FIRST}(Y_j)$  ( $1 \leq j \leq k$ )，则

$\epsilon \in \text{FIRST}(X)$




例. **G**:  $E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \varepsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \varepsilon$   
 $F \rightarrow (E) \mid i$

求每个非终结符号的**FIRST**集合

解:  $\text{FIRST}(E) = \text{FIRST}(T)$   
 $= \text{FIRST}(F)$   
 $= \{ (, i \}$

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

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



例.  $G: E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \varepsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \varepsilon$   
 $F \rightarrow (E) \mid i$

求每个产生式右部符号串的**FIRST**集合

解:

$\text{FIRST}(TE')$	$=$	$\{ (, i \}$
$\text{FIRST}(+TE')$	$=$	$\{ + \}$
$\text{FIRST}(FT')$	$=$	$\{ (, i \}$
$\text{FIRST}(*FT')$	$=$	$\{ * \}$
$\text{FIRST}((E))$	$=$	$\{ ( \}$
$\text{FIRST}(i)$	$=$	$\{ i \}$



例.  $E \rightarrow TE'$ ;  $E' \rightarrow +TE' \mid \varepsilon$ ;  $T \rightarrow FT'$ ;  $T' \rightarrow *FT' \mid \varepsilon$ ;  $F \rightarrow (E) \mid i$

$$\text{FIRST}(E) = \text{FIRST}(T) - \{\varepsilon\} \cup \text{FIRST}(E') - \{\varepsilon\}$$

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

$$\text{FIRST}(T) = \text{FIRST}(F) - \{\varepsilon\} \cup \text{FIRST}(T') - \{\varepsilon\}$$

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

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

$$\text{FIRST}(TE') = \text{FIRST}(T) - \{\varepsilon\} \cup \text{FIRST}(E') - \{\varepsilon\}$$

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

$$\text{FIRST}(FT') = \text{FIRST}(F) - \{\varepsilon\} \cup \text{FIRST}(T') - \{\varepsilon\}$$


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

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

# FOLLOW(A)构造

对于文法**G**的每个非终结符，构造**FOLLOW(A)**的方法是：

- (1) 若**A**为文法开始符号，置**#**于**FOLLOW(A)**中；
- (2) 若有产生式 **$B \rightarrow \alpha A \beta$** ，  
则将 **$FIRST(\beta) - \{\epsilon\}$** 加到**FOLLOW(A)**中；
- (3) 若有 **$B \rightarrow \alpha A$** 或 **$B \rightarrow \alpha A \beta$** ，且 **$\beta \xRightarrow{*} \epsilon$**   
则将**FOLLOW(B)**加到**FOLLOW(A)**中；
- (4) 反复使用以上规则，直至 **FOLLOW(A)**不再增大为止。



例.  $G: E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \varepsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \varepsilon$   
 $F \rightarrow (E) \mid i$

求每个非终结符号的**FOLLOW**集合

解:  $\text{FOLLOW}(E) = \{ \#, ) \}$

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

$\text{FOLLOW}(T) = \{ +, \#, ) \}$

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

$\text{FOLLOW}(F) = \{ *, +, \#, ) \}$

例.  $E \rightarrow TE'$ ;  $E' \rightarrow +TE' \mid \varepsilon$ ;  $T \rightarrow FT'$ ;  $T' \rightarrow *FT' \mid \varepsilon$ ;  $F \rightarrow (E) \mid i$

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

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

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

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

$\text{FOLLOW}(E') = \text{FOLLOW}(E) - \{ \varepsilon \} \cup \text{FIRST}(E') - \{ \varepsilon \} = \{ \#, ) \}$

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

$\text{FOLLOW}(T') = \text{FOLLOW}(T) - \{ \varepsilon \} \cup \text{FIRST}(T') - \{ \varepsilon \} = \{ +, \#, ) \}$

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



## 内容线索

- 语法分析器的功能
- 自上而下分析方法概述
- LL(1) 分析方法
- ✓ 递归下降分析程序
- 预测分析程序



# 递归下降分析程序

## ■ 条件

- 满足上述LL(1)文法的条件

## ■ 构成

- 一组递归过程
- 每个递归过程对应G的一个非终结符

## ■ 基本思想

- 从文法开始符号出发，在语法规则(文法产生式)的支配下进行语法分析。逐个扫描源程序中的字符(单词符号)，根据文法和当前输入字符分析到下一个语法成分A时，便调用识别和分析A的子程序(或其自身)，如此继续下去。

## 程序形式

- (1) 对每一个非终结符 $A$ ，编写一个相应的子程序 $P(A)$ ;
- (2) 对于规则 $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$  相应的子程序 $P(A)$ 构造如下:

IF ch IN FIRST( $\alpha_1$ ) THEN  $P(\alpha_1)$

ELSE IF ch IN FIRST( $\alpha_2$ ) THEN  $P(\alpha_2)$

ELSE .....

ELSE IF ch IN FIRST( $\alpha_n$ ) THEN  $P(\alpha_n)$

ELSE IF ( $\epsilon \in \text{FIRST}(A)$ ) AND (ch IN FOLLOW( $A$ ))

THEN RETURN

ELSE ERROR



(3) 对于符号串  $\alpha = y_1 y_2 y_3 \dots y_m$ ，相应的子程序  $P(\alpha)$  为：

```
BEGIN    P (y1)  
          P (y2)  
          ...  
          P (ym)  
END
```

■ 如果  $y_i \in V_T$ ，则  $P(y_i)$  为：

```
IF ch= yi THEN read(ch)  
ELSE ERROR;
```

■ 如果  $y_i \in V_N$ ，则  $P(y_i)$  为上述 (2) 中相应的子程序



例.  $E \rightarrow TE'$ ;  $E' \rightarrow +TE' \mid \varepsilon$ ;  $T \rightarrow FT'$ ;  $T' \rightarrow *FT' \mid \varepsilon$ ;  $F \rightarrow (E) \mid i$

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

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

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

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

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

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

**$E \rightarrow TE'$**

**$P(E);$**

**Begin**

**$P(T); P(E');$**

**End;**

**$E' \rightarrow +TE' \mid \varepsilon$**

**$P(E');$**

**Begin**

**If  $ch = '+'$  Then**

**begin**

**read(ch);**

**$P(T); P(E');$**

**End;**

**//  $ch \in \text{Follow}(E')$ ?**

**End;**

例.  $E \rightarrow TE'$ ;  $E' \rightarrow +TE' \mid \varepsilon$ ;  $T \rightarrow FT'$ ;  $T' \rightarrow *FT' \mid \varepsilon$ ;  $F \rightarrow (E) \mid i$

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

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

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

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

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

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

$T \rightarrow FT'$

P( T );

Begin

    P( F ); P( T' );

End;

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

P( T' );

Begin

    If ch='\*' Then

        begin

            read(ch);

            P( F ); P( T' );

        End;

        // **ch**  $\in$  **Follow(T')**?

End;

例.  $E \rightarrow TE'$ ;  $E' \rightarrow +TE' \mid \varepsilon$ ;  $T \rightarrow FT'$ ;  $T' \rightarrow *FT' \mid \varepsilon$ ;  $F \rightarrow (E) \mid i$

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

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

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

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

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

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

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

**P( F );**

**Begin**

**if** ch='i' **then** read(ch)

**else if** ch='(' **then**

**begin**

            read(ch); P( E );

**if** ch=')' **then**

                read(ch)

**else** Error

**End**

**else** Error;

**End;**

例.  $E \rightarrow TE'$ ;  $E' \rightarrow +TE' \mid \varepsilon$ ;  $T \rightarrow FT'$ ;  $T' \rightarrow *FT' \mid \varepsilon$ ;  $F \rightarrow (E) \mid i$

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

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

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

```
P( E )
BEGIN
  P(T); P(E')
END;
```

```
P(E')
IF ch = " + " THEN
  BEGIN
    read(ch); P(T); P(E');
  END;
ELSE IF ch = " )" THEN
  return;
ELSE IF ch = " #" THEN
  return;
ELSE ERROR;
```

```
P( T )
BEGIN
  P(F); P(T')
END;
```

```
P(T')
IF ch = " *" THEN
  BEGIN
    read(ch); P(F); P(T');
  END;
ELSE IF (ch = " + " OR
ch = " )" OR ch = " # ") THEN
  return;
ELSE ERROR;
```

```
P( F )
IF ch = 'i' THEN read(ch);
ELSE IF ch = '(' THEN
  BEGIN
    read(ch); P(E);
    IF ch = ')' THEN read(ch);
    ELSE ERROR
  END
ELSE ERROR;
```



## 内容线索

- 语法分析器的功能
- 自上而下分析方法概述
- LL(1) 分析方法
- 递归下降分析程序
- ✓ 预测分析程序



# 预测分析程序

## ■ 递归下降分析器的局限性

- 需要具有能够实现递归过程的语言和编译系统

## ■ 预测分析程序

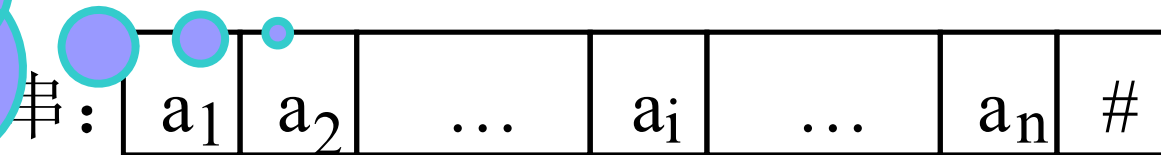
- 使用一个分析表和符号栈进行联合控制，是实现LL(1)分析的另一种有效方法。



## 预测分析程序基本思想

- 根据输入串的当前输入符确定选用某一个产生式进行推导，当该输入符与推导的第一个符号相同时，再取输入串的下一个符号，继续确定下一个推导应选的产生式，如此下去，直到推出被分析的输入串为止。
- 预测分析程序（**LL(1)**）分析器组成
  - **LL(1)**分析表（预测分析表）
  - 符号栈（后进先出）
  - 控制程序（表驱动程序）组成。

待分析的符号串,它以“#”作为结束标志。



表项:  $M[A, a]$

$A$ 的产生式

存放文法符号。  
分析开始时栈底为“#”

预测分析程序



## LL (1) 分析表

- 若文法有  $m$  个非终结符  $n$  个终结符，则 LL (1) 分析表是一个  $(m+1)*(n+2)$  的矩阵  $M$ 
  - 行标题为文法非终结符
  - 列标题为终结符号和输入结束符 #
  - $M[A, a]$  为一条关于  $A$  的产生式，指出当  $A$  面临  $a$  时，应使用的产生式或空格(出错标志)

例.  $E \rightarrow TE'$ ;  $E' \rightarrow +TE' \mid \varepsilon$ ;

$T \rightarrow FT'$ ;  $T' \rightarrow *FT' \mid \varepsilon$ ;

$F \rightarrow (E) \mid i$

LL(1)分析表

	i	+	*	(	)	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow i$			$F \rightarrow (E)$		

# LL(1)分析表的构造

- 预测分析程序中除分析表因文法的不同而不同外，分析栈、控制程序都相同。因此构造一个预测分析程序实际上就是构造文法的LL(1)分析表。

- 问题

1) 把产生式填到何处?

2) 按  $A \xRightarrow{*} ?$  将产生式分为两种:

一种是:  $A \xRightarrow{*} a...$

另种是:  $A \xRightarrow{*} \epsilon$



## 分析表构造算法

- (1) 对每个产生式  $A \rightarrow \alpha$ , 执行 (2) 和 (3)
- (2) 若  $a \in \text{FIRST}(\alpha)$ , 置  $M[A, a] = A \rightarrow \alpha$
- (3) 若  $\epsilon \in \text{FIRST}(\alpha)$ , 对  $b \in \text{FOLLOW}(A)$  置  $M[A, b] = A \rightarrow \epsilon$
- (4) 其余置  $M[A, a] = \text{ERROR}$

例.  $E \rightarrow TE'$ ;  $E' \rightarrow +TE' \mid \varepsilon$ ;  $T \rightarrow FT'$ ;  $T' \rightarrow *FT' \mid \varepsilon$ ;  $F \rightarrow (E)i$

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

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

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

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

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

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

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

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

	i	+	*	(	)	#
E						
E'						
T						
T'						
F						



# 栈

- 栈 **STACK**存放分析过程中的文法符号
  - 分析开始时栈底先放一个 “#”，再压入文法开始符；当分析栈中仅剩 “#”且输入串指针指向串尾的 “#”时，分析成功。

# 总控程序

总控程序根据栈顶符号 $x$ 和当前输入符 $a$ ，查表决定分析器的动作

- (1) 若 $x = a = \text{"\#"}$ ，即STACK 栈顶符号为“#”，当前输入符号为“#”，则分析成功。
- (2) 若 $x = a \neq \text{"\#"}$ ，即栈顶符号 $x$ 与当前输入符 $a$ 匹配，则将 $x$ 从栈顶弹出，输入串指针后移，读入下一个符号存入 $a$ ，继续对下一个字符进行分析。
- (3) 若 $x$ 为非终结符 $A$ ，则查分析表 $M[A, a]$ :
  - 1) 若 $M[A, a]$ 为一产生式，则 $A$ 自栈顶弹出， $M[A, a]$ 中产生式的右部符号逆序压栈；
  - 2) 若 $M[A, a]$ 为 $A \rightarrow \epsilon$ ，则只将 $A$ 自栈顶弹出。
  - 3) 若 $M[A, a]$ 为空，则发现语法错误，调用出错处理程序进行处理。

## 总控程序的伪码描述

**BEGIN**

# 及S 进栈(push('#');push('S'));

把第一个输入符读入a;

**FLAG := TRUE;**

**WHILE FLAG DO**

**BEGIN**

把STACK栈顶弹出放在X中( $X = \text{pop}()$ );

**IF  $X \in V_T$  THEN**

**IF  $X = a$  THEN 将下一输入符读入a ELSE ERROR**

**ELSE IF  $X = \text{"\#"} THEN$**

**IF  $X = a$  THEN FLAG := FALSE ELSE ERROR**

**ELSE IF  $M[X, a] = \{X \rightarrow X_1 \dots X_k\}$**

**THEN 把 $X_k, X_{k-1}, \dots, X_1$  逐一进栈**

**ELSE ERROR**

**END OF WHILE;**

**END**





**i + i \* i #**  
↑

T
E'
#

	i	+	*	(	)	#
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ε	E' → ε
T	T → FT'			T → FT'		
T'		T' → ε	T' → *FT'		T' → ε	T' → ε
F	F → i			F → (E)		

栈

输入

产生式

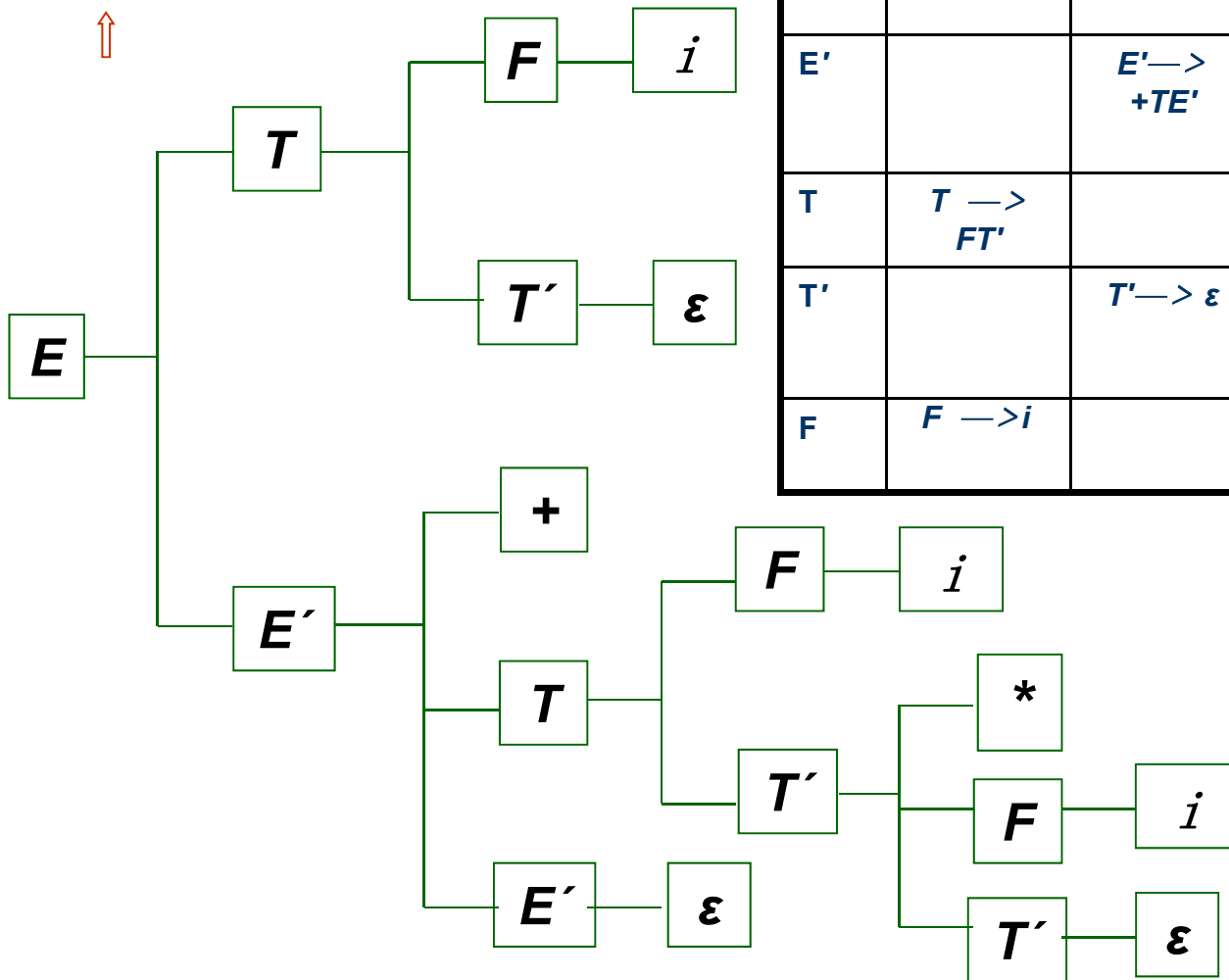
0	#E	i+i*i#	
1	#E'T	i+i*i#	$E \rightarrow TE'$
2	#E'T'F	i+i*i#	$T \rightarrow FT'$
3	#E'T'i	i+i*i#	$F \rightarrow i$
4	#E'T'	+i*i#	
5	#E'	+i*i#	$T' \rightarrow \varepsilon$
6	#E'T'+	+i*i#	$E' \rightarrow +TE'$
7	#E'T'	i*i#	
8	#E'T'F	i*i#	$T \rightarrow FT'$
9	#E'T'i	i*i#	$F \rightarrow i$
10	#E'T'	*i#	
11	#E'T'F*	*i#	$T' \rightarrow *FT'$
12	#E'T'F	i#	
13	#E'T'i	i#	$F \rightarrow i$
14	#E'T'	#	
15	#E'	#	$T' \rightarrow \varepsilon$
16	#	#	$E' \rightarrow \varepsilon$

## 结论

- 输出的产生式就是最左推导的产生式。栈中存放产生式右部，等待与 $a$ 匹配；
- 当栈顶 $X = a$ 时，分析表指出如何扩充语法树，出错能马上发现。
- 实质：
  - 栈：部分句型，句型右部，还未得到推导的符号。
  - 表：指出 $V_N$ 按哪一条扩充，依赖于 $V_T$

✓ 上述分析过程生成的语法树:

$i + i * i \# :$



	i	+	*	(	)	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow i$			$F \rightarrow (E)$		

## 分析表与LL(1) 文法

- 若文法**G**的预测分析表  $M[A, a]$  不含有多重定义入口，当且仅当**G**为LL(1)文法。
- 文法**G**是LL(1)的，则对于**G**的每一个非终结符**A**的任何两个不同产生式  $A \rightarrow \alpha \mid \beta$ ,有:
  - (1)  $FIRST(\alpha) \cap FIRST(\beta) = \varphi$
  - (2) 若  $\beta \xRightarrow{*} \varepsilon$ , 则  $FIRST(\alpha) \cap FOLLOW(A) = \Phi$



# 作业

## ■ P81

☐ 1

☐ 2