



编译原理

第四章

语法分析

哈尔滨工业大学 朱庆福

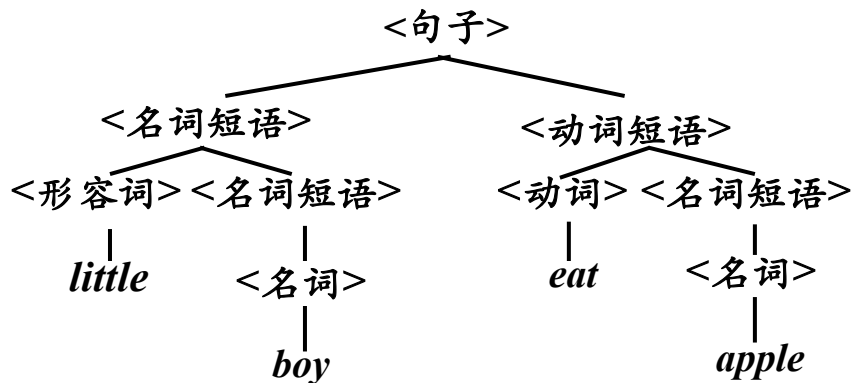


引言

➤ 语法分析的主要任务

- 根据给定的**文法**，识别**输入句子**的各个成分，从而构造出句子的**分析树**
- 大部分程序设计语言的**语法构造**可以用**CFG**来描述，**CFG**以 **token** 作为**终结符**
- 大部分语法分析器都期望文法是**无二义性**的，否则，就不能为一个句子构造**唯一**的语法分析树

语法分析的种类




从左向右扫描输入，
每次扫描一个符号

➤ 自顶向下的分析(*Top-Down Parsing*)

- 从分析树的顶部（根节点）向底部（叶节点）构造分析树
- 从文法开始符号 S 推导出串 w

➤ 自底向上的分析(*Bottom-up Parsing*)

- 从分析树的底部（叶节点）向顶部（根节点）构造分析树
- 将一个串 w 归约为文法开始符号 S

- 
- 最高效的自顶向下和自底向上方法只能处理某些文法子类，但是其中的某些子类，特别是 *LL* 和 *LR* 文法，其表达能力足以描述现代程序设计语言的大部分语法构造



本章内容

4.1 自顶向下的分析

4.2 预测分析法

4.3 自底向上的分析

4.4 LR分析法

4.5 语法分析器自动生成工具

4.1 自顶向下的分析 (*Top-Down Parsing*)

- 从分析树的顶部（根节点）向底部（叶节点）方向构造分析树
- 可以看成是从文法开始符号 S 推导出词串 w 的过程

➤ 例

文法

① $E \rightarrow E + E$

② $E \rightarrow E * E$

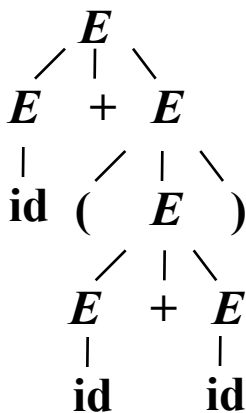
③ $E \rightarrow (E)$

④ $E \rightarrow \text{id}$

输入

$\text{id} + (\text{id} + \text{id})$

分析树:



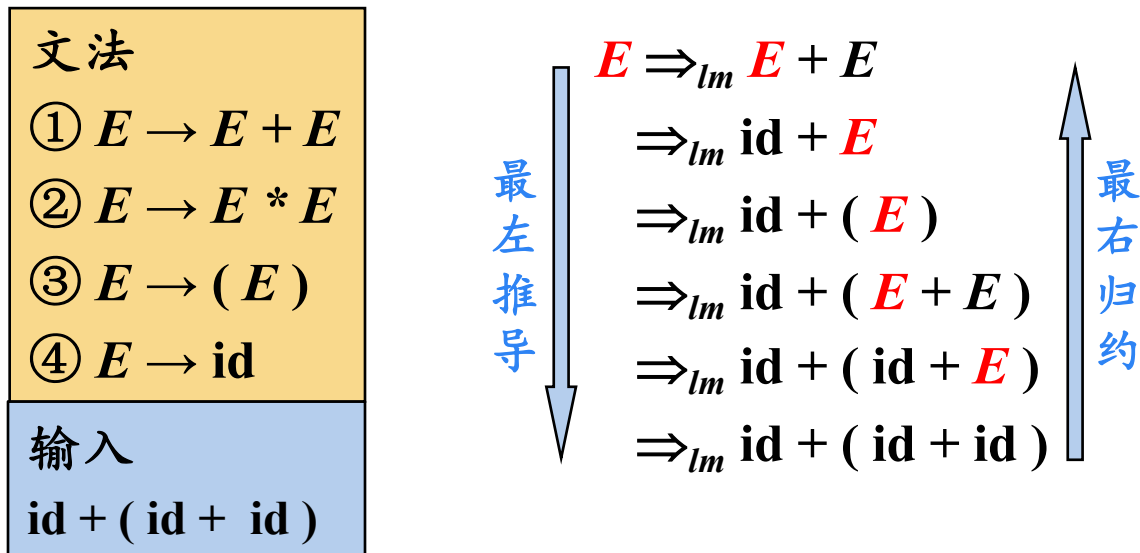
推导: $E \Rightarrow E + E$
 $\Rightarrow \text{id} + E$
 $\Rightarrow \text{id} + (E)$
 $\Rightarrow \text{id} + (E + E)$
 $\Rightarrow \text{id} + (\text{id} + E)$
 $\Rightarrow \text{id} + (\text{id} + \text{id})$

- 推导的每一步，都需要做两个选择
 - 替换当前句型中的哪个非终结符
 - 用该非终结符的哪个候选式进行替换

最左推导 (Left-most Derivation)

➤ 在**最左推导**中，总是选择每个句型的最左非终结符进行替换

➤ 例



➤ 如果 $S \Rightarrow_{lm}^* \alpha$ ，则称 α 是当前文法的最左句型 (leftmost-sentential form)

最右推导 (Right-most Derivation)

最右句型也称为规范句型

➤ 在**最右推导**中，总是选择每个句型的最右非终结符进行替换

➤ 例

文法
① $E \rightarrow E + E$
② $E \rightarrow E * E$
③ $E \rightarrow (E)$
④ $E \rightarrow \text{id}$
输入
$\text{id} + (\text{id} + \text{id})$

$$\begin{array}{l} E \Rightarrow_{rm} E + E \\ \Rightarrow_{rm} E + (E) \\ \Rightarrow_{rm} E + (E + E) \\ \Rightarrow_{rm} E + (E + \text{id}) \\ \Rightarrow_{rm} E + (\text{id} + \text{id}) \\ \Rightarrow_{rm} \text{id} + (\text{id} + \text{id}) \end{array}$$

最右推导 (Left arrow) 最左归约 (Right arrow)

➤ 如果 $S \Rightarrow_{rm}^* \alpha$ ，则称 α 是当前文法的最右句型 (rightmost-sentential form)

➤ 在自底向上的分析中，总是采用最左归约的方式，因此把最左归约称为规范归约，而最右推导相应地称为规范推导

最左推导和最右推导的唯一性

$$E \Rightarrow E + E$$

$$\Rightarrow E + (E)$$

$$\Rightarrow E + (E + E)$$

$$\Rightarrow E + (\text{id} + E)$$

$$\Rightarrow \text{id} + (\text{id} + E)$$

$$\Rightarrow \text{id} + (\text{id} + \text{id})$$

$$E \Rightarrow E + E$$

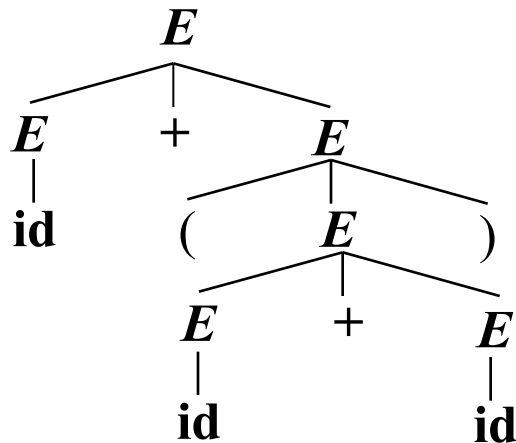
$$\Rightarrow \text{id} + E$$

$$\Rightarrow \text{id} + (E)$$

$$\Rightarrow \text{id} + (E + E)$$

$$\Rightarrow \text{id} + (E + \text{id})$$

$$\Rightarrow \text{id} + (\text{id} + \text{id})$$



$$E \Rightarrow_{lm} E + E$$

$$\Rightarrow_{lm} \text{id} + E$$

$$\Rightarrow_{lm} \text{id} + (E)$$

$$\Rightarrow_{lm} \text{id} + (E + E)$$

$$\Rightarrow_{lm} \text{id} + (\text{id} + E)$$

$$\Rightarrow_{lm} \text{id} + (\text{id} + \text{id})$$

$$E \Rightarrow_{rm} E + E$$

$$\Rightarrow_{rm} E + (E)$$

$$\Rightarrow_{rm} E + (E + E)$$

$$\Rightarrow_{rm} E + (E + \text{id})$$

$$\Rightarrow_{rm} E + (\text{id} + \text{id})$$

$$\Rightarrow_{rm} \text{id} + (\text{id} + \text{id})$$

自顶向下的语法分析采用最左推导方式

- 总是选择每个句型的最左非终结符进行替换
- 根据输入流中的当前终结符，选择最左非终结符的一个候选式

例

➤ 文法

$$\textcircled{1} E \rightarrow T E'$$

$$\textcircled{2} E' \rightarrow + T E' \mid \varepsilon$$

$$\textcircled{3} T \rightarrow F T'$$

$$\textcircled{4} T' \rightarrow * F T' \mid \varepsilon$$

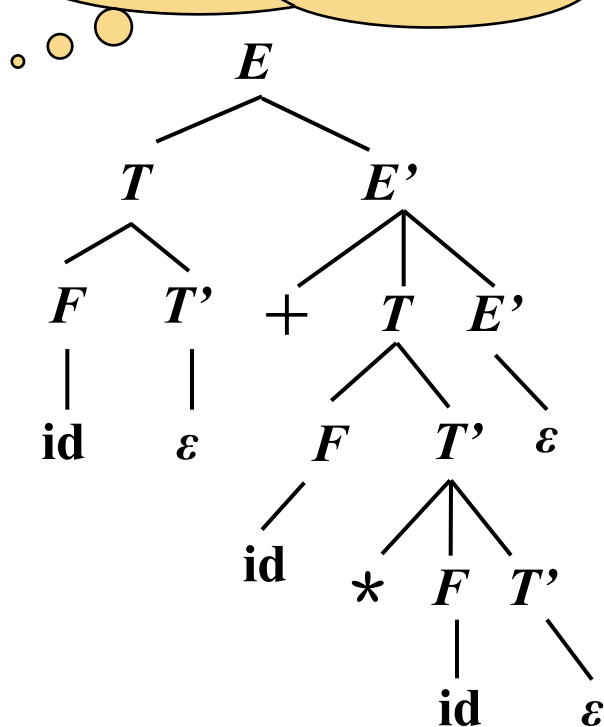
$$\textcircled{5} F \rightarrow (E) \mid \text{id}$$

➤ 输入

id + id * id



计算机是如何自动地实现这个自顶向下的分析过程的?



自顶向下语法分析的通用形式

➤ 递归下降分析 (Recursive-Descent Parsing)

- 由一组过程组成，每个过程对应文法的一个非终结符
- 从文法开始符号 S 对应的过程开始，其中递归调用文法中其它非终结符对应的过程。如果 S 对应的过程体恰好扫描了整个输入串，则成功完成语法分析

void $S()$

{ 选择一个 S 产生式: $S \rightarrow X_1 X_2 \dots X_n$;

for ($i = 1$ to n)

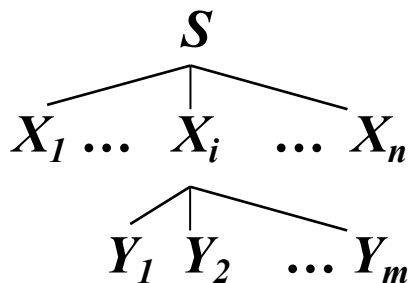
最左推导

$\left\{ \begin{array}{l} \text{if } X_i \in V_T \left\{ \begin{array}{l} \text{if } X_i == \text{Token} \text{ then } \text{GetNextToken}() \\ \text{else } (X_i \neq \text{Token}) \quad \text{Error}() \end{array} \right. \\ \text{else } (X_i \in V_N) \quad X_i() \end{array} \right.$

$X_i \rightarrow Y_1 Y_2 \dots Y_m$

}

自顶向下



自顶向下语法分析的通用形式

➤ 递归下降分析 (*Recursive-Descent Parsing*)

- 由一组过程组成，每个过程对应文法的一个非终结符
- 从文法开始符号 S 对应的过程开始，其中递归调用文法中其它非终结符对应的过程。如果 S 对应的过程体恰好扫描了整个输入串，则成功完成语法分析

```
void A() {  
1)  选择一个 $A$ 产生式,  $A \rightarrow X_1 X_2 \dots X_k$  ;  
2)  for ( $i = 1$  to  $k$ ) {  
3)      if ( $X_i$  是一个非终结符号)  
4)          调用过程  $X_i()$  ;  
5)      else if ( $X_i$  等于当前的输入符号  $a$ )  
6)          读入下一个输入符号;  
7)      else /* 发生了一个错误 */;  
    }  
}
```

自顶向下分析存在的问题

➤ 问题1

➤ 例：文法 G 同一非终结符的多个候选式存在共同前缀，将导致回溯现象

$$S \rightarrow aAd \mid aBe$$

$$A \rightarrow c$$

$$B \rightarrow b$$

➤ 输入

$a b c$



问题2

➤ 例

➤ 文法 G

$$E \rightarrow E + T \mid E - T \mid T$$

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

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

$$E \Rightarrow E + T$$

$$\Rightarrow E + T + T$$

$$\Rightarrow E + T + T + T$$

$$\Rightarrow \dots$$

➤ 输入

id + id * id



左递归文法会使递归下降分析器陷入无限循环

如果一个文法中有一个非终结符 A 使得对某个串 α 存在一个推导 $A \Rightarrow^+ A\alpha$ ，那么这个文法就是左递归的

含有 $A \rightarrow A\alpha$ 形式产生式的文法称为是直接左递归的 (immediate left recursive)

经过两步或两步以上推导产生的左递归称为是间接左递归的

消除直接左递归

$$A \rightarrow A\alpha \mid \beta (\alpha \neq \varepsilon, \beta \text{ 不以 } A \text{ 开头}) \quad r = \beta\alpha^*$$



$$A \rightarrow \beta A'$$

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

事实上，这种消除过程就是把左递归转换成了右递归

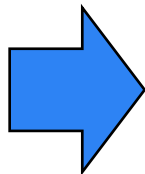
$$\begin{aligned} A &\Rightarrow A\alpha \\ &\Rightarrow A\alpha\alpha \\ &\Rightarrow A\alpha\alpha\alpha \\ &\dots \\ &\Rightarrow A\alpha \dots \alpha \\ &\Rightarrow \beta \alpha \dots \alpha \end{aligned}$$

➤ 例 $E \rightarrow E + T \mid T$

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

α β

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



$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \varepsilon$$

$$T \rightarrow F T'$$

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

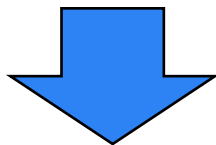
$$F \rightarrow (E) \mid \text{id}$$

$$\begin{aligned} A' &\Rightarrow \alpha A' \\ &\Rightarrow \alpha\alpha A' \\ &\Rightarrow \alpha\alpha\alpha A' \\ &\dots \\ &\Rightarrow \alpha \dots \alpha A' \\ &\Rightarrow \alpha \dots \alpha \end{aligned}$$

消除直接左递归的一般形式

$$A \rightarrow A \alpha_1 \mid A \alpha_2 \mid \dots \mid A \alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

($\alpha_i \neq \varepsilon$, β_j 不以 A 开头)



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

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

消除左递归是要付出代价的——引进了一些非终结符和 ε 产生式

消除间接左递归

➤ 例

$$S \rightarrow A a \mid b$$

$$\begin{aligned} S &\Rightarrow Aa \\ &\Rightarrow Sda \end{aligned}$$

$$A \rightarrow A c \mid S d \mid \varepsilon$$

➤ 将 S 的定义代入 A -产生式，得：

$$A \rightarrow A c \mid A a d \mid b d \mid \varepsilon$$

➤ 消除 A -产生式的直接左递归，得：

$$A \rightarrow b d A' \mid A'$$

$$A' \rightarrow c A' \mid a d A' \mid \varepsilon$$

消除左递归算法

- 输入：不含循环推导（即形如 $A \Rightarrow^+ A$ 的推导）和 ε -产生式的文法 G
- 输出：等价的无左递归文法
- 方法：

- 1) 按照某个顺序将非终结符号排序为 A_1, A_2, \dots, A_n .
- 2) for (从1到n的每个 i) {
- 3) for (从1到 $i-1$ 的每个 j) {
- 4) 将每个形如 $A_i \rightarrow A_j \gamma$ 的产生式替换为产生式组 $A_i \rightarrow \delta_1 \gamma | \delta_2 \gamma | \dots | \delta_k \gamma$,
 其中 $A_j \rightarrow \delta_1 | \delta_2 | \dots | \delta_k$, 是所有的 A_j 产生式
- 5) }
- 6) 消除 A_i 产生式之间的直接左递归
- 7) }

提取左公因子 (*Left Factoring*)

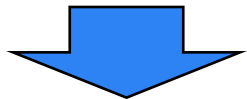
➤ 例

➤ 文法 G

➤ $S \rightarrow aAd \mid aBe$

➤ $A \rightarrow c$

➤ $B \rightarrow b$



➤ 文法 G'

➤ $S \rightarrow a S'$

➤ $S' \rightarrow Ad \mid Be$

➤ $A \rightarrow c$

➤ $B \rightarrow b$

通过改写产生式来推迟决定，
等读入了足够多的输入，获得
足够信息后再做出正确的选择

提取左公因子算法

- 输入：文法 G
- 输出：等价的提取了左公因子的文法
- 方法：

对于每个非终结符 A ，找出它的两个或多个选项之间的最长公共前缀 α 。如果 $\alpha \neq \varepsilon$ ，即存在一个非平凡的(*nontrivial*)公共前缀，那么将所有 A -产生式

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

替换为

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

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

其中， γ_i 表示所有不以 α 开头的产生式体； A' 是一个新的非终结符。不断应用这个转换，直到每个非终结符的任意两个产生式体都没有公共前缀为止

自顶向下语法分析的通用形式

➤ 递归下降分析 (*Recursive-Descent Parsing*)

- 由一组过程组成，每个过程对应一个非终结符
- 从文法开始符号 S 对应的过程开始，其中递归调用文法中其它非终结符对应的过程。如果 S 对应的过程体恰好扫描了整个输入串，则成功完成语法分析

```
void A() {  
1)  选择一个 $A$ 产生式,  $A \rightarrow X_1 X_2 \dots X_k$  ;  
2)  for ( $i = 1$  to  $k$ ) {  
3)      if ( $X_i$  是一个非终结符号)  
4)          调用过程  $X_i()$  ;  
5)      else if ( $X_i$  等于当前的输入符号  $a$ )  
6)          读入下一个输入符号;  
7)      else /* 发生了一个错误 */;  
    }  
}
```

可能需要回溯(backtracking),
导致效率较低

需要回溯的分析 (不确定的分析)

预测分析 (*Predictive Parsing*)

- 预测分析是递归下降分析技术的一个特例，通过在输入中向前看固定个数（通常是一个）符号来选择正确的 A -产生式。
- 可以对某些文法构造出向前看 k 个输入符号的预测分析器，该类文法有时也称为 $LL(k)$ 文法类
- 预测分析不需要回溯，是一种确定的自顶向下分析方法



提纲

4.1 自顶向下的分析

4.2 预测分析法

4.3 自底向上的分析

4.4 LR分析法

4.5 语法分析器自动生成工具

4.2 预测分析法

- 4.2.1 LL(1) 文法
- 4.2.2 递归的预测分析法
- 4.2.3 非递归的预测分析法
- 4.2.4 预测分析中的错误恢复

4.2.1 LL(1) 文法

假如允许 S -文法包含 ϵ 产生式，
将会产生什么问题？

➤ 预测分析法的工作过程

➤ 从文法开始符号出发，在每一步推导过程中根据当前句型的**最左非终结符** A 和当前**输入符号** a ，选择正确的 A -产生式。为保证分析的**确定性**，选出的候选式必须是**唯一的**。

➤ S -文法（简单的确定性文法，*Korenjak & Hopcroft*, 1966）

每个产生式的右部都以**终结符**开始

同一非终结符的各个候选式的**首终结符**都不同

S -文法**不含 ϵ 产生式**

例

<ul style="list-style-type: none">① $S \rightarrow aBCD$② $B \rightarrow bC$③ $B \rightarrow dB$④ $B \rightarrow \varepsilon$⑤ $C \rightarrow c$⑥ $C \rightarrow a$⑦ $D \rightarrow e$	➤ 输入	$a \ d \ a \ e$	$a \ d \ e \ e$
	➤ 推导	$\begin{aligned} &S \\ \Rightarrow &aBCD \\ \Rightarrow &adBCD \\ \Rightarrow &adCD \\ \Rightarrow &adaD \\ \Rightarrow &adae \end{aligned}$	$\begin{aligned} &S \\ \Rightarrow &aBCD \\ \Rightarrow &adBCD \\ \Rightarrow &adCD \end{aligned}$

可以紧跟 B 后面出现的终结符: c 、 a

➤ 什么时候使用 ε 产生式?

- 如果当前某非终结符 A 与当前输入符 a 不匹配时, 若存在 $A \rightarrow \varepsilon$, 可以通过检查 a 是否可以出现在 A 的后面, 来决定是否使用产生式 $A \rightarrow \varepsilon$ (若文法中无 $A \rightarrow \varepsilon$, 则应报错)

非终结符的后继符号集

➤ 非终结符 A 的后继符号集

➤ 可能在某个句型中紧跟在 A 后边的终结符 a 的集合，记为 $FOLLOW(A)$

$$FOLLOW(A) = \{a \mid S \Rightarrow^* \alpha A a \beta, a \in V_T, \alpha, \beta \in (V_T \cup V_N)^*\}$$

例

- (1) $S \rightarrow aBCD$
- (2) $B \rightarrow bC$ ← b 输入
- (3) $B \rightarrow dB$ ← d
- (4) $B \rightarrow \varepsilon$ ← $\{a, c\}$
- (5) $C \rightarrow c$
- (6) $C \rightarrow a$
- (7) $D \rightarrow e$

$$FOLLOW(B) = \{a, c\}$$

如果 A 是某个句型的的最右符号，则将结束符“\$”添加到 $FOLLOW(A)$ 中

产生式的可选集

➤ 产生式 $A \rightarrow \beta$ 的 **可选集** 是指可以选用该产生式进行推导时对应的 **输入符号** 的集合，记为 $SELECT(A \rightarrow \beta)$

➤ $SELECT(A \rightarrow a\beta) = \{a\}$

➤ $SELECT(A \rightarrow \varepsilon) = FOLLOW(A)$

➤ q _文法

➤ 每个产生式的右部或为 ε ，或以 **终结符** 开始

➤ 具有相同左部的产生式有 **不相交的可选集**

q _文法不含右部以 **非终结符** 打头的产生式

串首终结符集

➤ 串首终结符

- 串首第一个符号，并且是终结符。简称首终结符
- 给定一个文法符号串 α ， α 的串首终结符集 $FIRST(\alpha)$ 被定义为可以从 α 推导出的所有串首终结符构成的集合。如果 $\alpha \Rightarrow^* \varepsilon$ ，那么 ε 也在 $FIRST(\alpha)$ 中
- 对于 $\forall \alpha \in (V_T \cup V_N)^+$, $FIRST(\alpha) = \{ a \mid \alpha \Rightarrow^* a\beta, a \in V_T, \beta \in (V_T \cup V_N)^* \}$;
- 如果 $\alpha \Rightarrow^* \varepsilon$, 那么 $\varepsilon \in FIRST(\alpha)$

FIRST(α)的计算

$$\alpha = X_1 X_2 \cdots X_k$$

➤ **FIRST(α)**: 首终结符集。 α 能够推出的**所有**终结符串中位于**串首**的那些**终结符**构成的集合

$$= FIRST(X_1) \cup FIRST(X_2) \cup FIRST(X_3) \cup \cdots$$

if $X_1 \Rightarrow^* \varepsilon$ if $X_2 \Rightarrow^* \varepsilon$

$$\alpha \Rightarrow^* \varepsilon \Leftrightarrow \varepsilon \in FIRST(\alpha)$$

$$FIRST(X_i) = \begin{cases} \{X_i\} & \text{if } X_i \in V_T \\ \text{通过 } X_i \text{-产生式右部求} & \text{if } X_i \in V_N \end{cases}$$

计算文法符号 X 的 $FIRST(X)$

➤ $FIRST(X)$: 可以从 X 推导出的所有串首终结符构成的集合

➤ 如果 $X \Rightarrow^* \varepsilon$, 那么 $\varepsilon \in FIRST(X)$

➤ 例

$$\textcircled{1} \quad E \rightarrow TE' \qquad FIRST(E) = \{ (\quad \text{id} \quad \}$$

$$\textcircled{2} \quad E' \rightarrow +TE' \mid \varepsilon \qquad FIRST(E') = \{ + \quad \varepsilon \quad \}$$

$$\textcircled{3} \quad T \rightarrow FT' \qquad FIRST(T) = \{ (\quad \text{id} \quad \}$$

$$\textcircled{4} \quad T' \rightarrow *FT' \mid \varepsilon \qquad FIRST(T') = \{ * \quad \varepsilon \quad \}$$

$$\textcircled{5} \quad F \rightarrow (E) \mid \text{id} \qquad FIRST(F) = \{ (\quad \text{id} \quad \}$$

算法

- 不断应用下列规则，直到没有新的终结符或 ε 可以被加入到任何 $FIRST$ 集合中为止
 - 如果 X 是一个终结符，那么 $FIRST(X) = \{X\}$
 - 如果 X 是一个非终结符，且 $X \rightarrow Y_1 \dots Y_k \in P$ ($k \geq 1$)，那么如果对于某个 i ， a 在 $FIRST(Y_i)$ 中且 ε 在所有的 $FIRST(Y_1), \dots, FIRST(Y_{i-1})$ 中(即 $Y_1 \dots Y_{i-1} \Rightarrow^* \varepsilon$)，就把 a 加入到 $FIRST(X)$ 中。如果对于所有的 $j = 1, 2, \dots, k$ ， ε 在 $FIRST(Y_j)$ 中，那么将 ε 加入到 $FIRST(X)$
 - 如果 $X \rightarrow \varepsilon \in P$ ，那么将 ε 加入到 $FIRST(X)$ 中

计算串 $X_1X_2 \dots X_n$ 的 *FIRST* 集合

- 向 $FIRST(X_1X_2 \dots X_n)$ 加入 $FIRST(X_1)$ 中所有的非 ϵ 符号
- 如果 ϵ 在 $FIRST(X_1)$ 中，再加入 $FIRST(X_2)$ 中的所有非 ϵ 符号；
如果 ϵ 在 $FIRST(X_1)$ 和 $FIRST(X_2)$ 中，再加入 $FIRST(X_3)$ 中的所有非 ϵ 符号，以此类推
- 最后，如果对所有的 i ， ϵ 都在 $FIRST(X_i)$ 中，那么将 ϵ 加入到 $FIRST(X_1X_2 \dots X_n)$ 中

产生式 $A \rightarrow \alpha$ 的可选集

➤ 产生式 $A \rightarrow \alpha$ 的可选集 *SELECT*

➤ 如果 $\varepsilon \notin FIRST(\alpha)$, 那么 $SELECT(A \rightarrow \alpha) = FIRST(\alpha)$

➤ 如果 $\varepsilon \in FIRST(\alpha)$, 那么 $SELECT(A \rightarrow \alpha) = (FIRST(\alpha) - \{\varepsilon\}) \cup FOLLOW(A)$

LL(1)文法

- 文法 G 是 $LL(1)$ 的，当且仅当 G 的任意两个具有相同左部的产生式 $A \rightarrow \alpha \mid \beta$ 满足下面的条件：
 - 不存在终结符 a 使得 α 和 β 都能够推导出以 a 开头的串
 - α 和 β 至多有一个能推导出 ε
 - 如果 $\beta \Rightarrow^* \varepsilon$ ，则 $FIRST(\alpha) \cap FOLLOW(A) = \Phi$ ；
如果 $\alpha \Rightarrow^* \varepsilon$ ，则 $FIRST(\beta) \cap FOLLOW(A) = \Phi$ ；

同一非终结符的各个产生式的可选集互不相交

可以为 $LL(1)$ 文法构造预测分析器

LL(1)文法

- 文法 G 是 $LL(1)$ 的，当且仅当 G 的任意两个具有相同左部的产生式 $A \rightarrow \alpha \mid \beta$ 满足下面的条件：
 - 不存在终结符 a 使得 α 和 β 都能够推导出以 a 开头的串
 - α 和 β 至多有一个能推导出 ε
 - 如果 $\beta \Rightarrow^* \varepsilon$ ，则 $FIRST(\alpha) \cap FOLLOW(A) = \Phi$ ；
如果 $\alpha \Rightarrow^* \varepsilon$ ，则 $FIRST(\beta) \cap FOLLOW(A) = \Phi$ ；

- 第一个“ L ”表示从左（Left）向右扫描输入
- 第二个“ L ”表示产生最左（Left）推导
- “1”表示在每一步中只需要向前看一个输入符号来决定语法分析动作

计算非终结符 A 的 $FOLLOW(A)$

➤ $FOLLOW(A)$: 可能在某个句型中紧跟在 A 后边的终结符 a 的集合

$$FOLLOW(A) = \{a \mid S \Rightarrow^* \alpha A a \beta, a \in V_T, \alpha, \beta \in (V_T \cup V_N)^*\}$$

➤ 如果 A 是某个句型的最右符号, 则将结束符“\$”添加到 $FOLLOW(A)$ 中

例

① $E \rightarrow \underline{TE'}$ $FIRST(E) = \{ (\text{id} \}$ $FOLLOW(E) = \{ \$) \}$

② $E' \rightarrow \underline{+TE'} \mid \varepsilon$ $FIRST(E') = \{ + \varepsilon \}$ $FOLLOW(E') = \{ \$) \}$

③ $T \rightarrow \underline{FT'}$ $FIRST(T) = \{ (\text{id} \}$ $FOLLOW(T) = \{ + \$) \}$

④ $T' \rightarrow \underline{*FT'} \mid \varepsilon$ $FIRST(T') = \{ * \varepsilon \}$ $FOLLOW(T') = \{ + \$) \}$

⑤ $F \rightarrow \underline{(E)} \mid \text{id}$ $FIRST(F) = \{ (\text{id} \}$ $FOLLOW(F) = \{ * + \$) \}$

$FOLLOW(A)$ 计算方法

- 不断应用下列规则，直到没有新的终结符可以被加入到任何 $FOLLOW$ 集合中为止
- 将 $\$$ 放入 $FOLLOW(S)$ 中，其中 S 是开始符号， $\$$ 是输入右端的结束标记
- 如果存在一个产生式 $A \rightarrow \alpha B \beta$ ，那么 $FIRST(\beta)$ 中除 ε 之外的所有符号都在 $FOLLOW(B)$ 中
- 如果存在一个产生式 $A \rightarrow \alpha B$ ，或存在产生式 $A \rightarrow \alpha B \beta$ 且 $FIRST(\beta)$ 包含 ε ，那么 $FOLLOW(A)$ 中的所有符号都在 $FOLLOW(B)$ 中

例：表达式文法各产生式的 $SELECT$ 集

X	$FIRST(X)$	$FOLLOW(X)$
E	(id	\$)
E'	+ ϵ	\$)
T	(id	+) \$
T'	* ϵ	+) \$
F	(id	* +) \$

表达式文法是 $LL(1)$ 文法

- (1) $E \rightarrow T E'$ $SELECT(1) = \{ (\text{ id } \}$
- (2) $E' \rightarrow + T E'$ $SELECT(2) = \{ + \}$
- (3) $E' \rightarrow \epsilon$ $SELECT(3) = \{ \$) \}$
- (4) $T \rightarrow F T'$ $SELECT(4) = \{ (\text{ id } \}$
- (5) $T' \rightarrow * F T'$ $SELECT(5) = \{ * \}$
- (6) $T' \rightarrow \epsilon$ $SELECT(6) = \{ +) \$ \}$
- (7) $F \rightarrow (E)$ $SELECT(7) = \{ (\}$
- (8) $F \rightarrow \text{id}$ $SELECT(8) = \{ \text{id} \}$

预测分析表

	产生式	<i>SELECT</i>
<i>E</i>	$E \rightarrow TE'$	(id
<i>E'</i>	$E' \rightarrow +TE'$	+
	$E' \rightarrow \varepsilon$	\$)
<i>T</i>	$T \rightarrow FT'$	(id
<i>T'</i>	$T' \rightarrow *FT'$	*
<i>F</i>	$F \rightarrow (E)$	(
	$F \rightarrow id$	id

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

如何实现预测分析？

- 递归的方式： 基于预测分析表对递归下降分析法进行扩展
- 非递归的方式： 显式地维护一个栈结构来模拟最左推导过程

4.2.2 递归的预测分析法

➤ 是对递归下降分析框架的扩展

$A(\text{Token})$

{

if $\text{Token} \in \text{SELECT}(A \rightarrow \alpha_1)$

code_1 ;

if $\text{Token} \in \text{SELECT}(A \rightarrow \alpha_2)$

code_2 ;

...

if $\text{Token} \in \text{SELECT}(A \rightarrow \alpha_n)$

code_n ;

}

code_i :

设 $\alpha_i = X_1 X_2 \dots X_k$

for ($j = 1$ to k)

$\left[\begin{array}{l} \text{if } X_j \in V_T \quad \left[\begin{array}{l} \text{if } X_j == \text{Token} \quad \text{then GetNext}(\text{Token}) \\ \text{else } (X_j \neq \text{Token}) \quad \text{Error}() \end{array} \right. \\ \text{else } (X_j \in V_N) \quad X_j(\text{Token}) \end{array} \right.$

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

$\begin{array}{ccc} | & | & | \\ \text{code}_1 & & \text{code}_n \\ & | & \\ & \text{code}_2 & \end{array}$

例

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; \text{s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

```
program DESCENT;  
  begin  
    GETNEXT(TOKEN);  
    PROGRAM(TOKEN);  
    if TOKEN ≠ '$' then ERROR;  
  end
```

SELECT(4)={:}

SELECT(7)={end}

例

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle \text{ : } \langle \text{TYPE} \rangle \text{ ; } \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow \text{ , id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow \text{ ; s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

SELECT(4)={:}

SELECT(7)={end}

```
procedure PROGRAM(TOKEN);
begin
  → if TOKEN≠'program' then ERROR;
    GETNEXT(TOKEN);

  → DECLIST(TOKEN);

  → if TOKEN≠':' then ERROR;
    GETNEXT(TOKEN);

  → TYPE(TOKEN)

  → if TOKEN≠';' then ERROR;
    GETNEXT(TOKEN);

  → STLIST(TOKEN);

  → if TOKEN≠'end' then ERROR;
    GETNEXT(TOKEN);
end
```

例

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; \text{s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

SELECT(4)={:}
SELECT(7)={end}

```
procedure DECLIST(TOKEN);  
  begin  
    if TOKEN≠'id' then ERROR;  
    GETNEXT(TOKEN);  
    DECLISTN(TOKEN);  
  end
```

例

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; \text{s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

SELECT(4)={:}

SELECT(7)={end}

```
procedure DECLISTN(TOKEN);  
  begin  
    if TOKEN = ',' then  
      begin  
        GETNEXT(TOKEN);  
  
        if TOKEN ≠ 'id' then ERROR;  
        GETNEXT(TOKEN);  
  
        DECLISTN(TOKEN);  
      end  
    else if TOKEN ≠ ':' then ERROR;  
  end
```

例

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow s \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; s \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

SELECT(4)={:}

SELECT(7)={end}

```
procedure STLIST(TOKEN);  
  begin  
    if TOKEN≠'s' then ERROR;  
    GETNEXT(TOKEN);  
    STLISTN(TOKEN);  
  end
```


例

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; \text{s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

SELECT(4)={:}

SELECT(7)={end}

```
procedure STLISTN(TOKEN);  
  begin  
    if TOKEN = ';' then  
      begin  
        GETNEXT(TOKEN);  
  
        if TOKEN ≠ 's' then ERROR;  
        GETNEXT(TOKEN);  
  
        STLISTN(TOKEN);  
      end  
    else if TOKEN ≠ 'end' then ERROR;  
  end
```

例

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; \text{s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

SELECT(4)={:}
SELECT(7)={end}

```
procedure TYPE(TOKEN);  
  begin  
    if TOKEN≠'real' and TOKEN≠'int'  
      then ERROR;  
    GETNEXT(TOKEN);  
  end
```

例 (MOOC)

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; \text{s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

SELECT(4)={:}

SELECT(7)={end}

```
procedure PROGRAM(TOKEN);
begin
    if TOKEN≠'program' then ERROR;

    GETNEXT(TOKEN);
    DECLIST(TOKEN);

    if TOKEN≠':' then ERROR;

    GETNEXT(TOKEN);
    TYPE(TOKEN);

    GETNEXT(TOKEN);
    if TOKEN≠';' then ERROR;

    GETNEXT(TOKEN);
    STLIST(TOKEN);

    if TOKEN≠'end' then ERROR;
end
```

例 (MOOC)

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; \text{s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

SELECT(4)={:}
SELECT(7)={end}

```
procedure TYPE(TOKEN);  
  begin  
    if TOKEN≠'real' and TOKEN≠'int'  
      then ERROR;  
  end
```

例 (MOOC)

(1) $\langle \text{PROGRAM} \rangle \rightarrow \text{program } \langle \text{DECLIST} \rangle : \langle \text{TYPE} \rangle ; \langle \text{STLIST} \rangle \text{ end}$

(2) $\langle \text{DECLIST} \rangle \rightarrow \text{id } \langle \text{DECLISTN} \rangle$

(3) $\langle \text{DECLISTN} \rangle \rightarrow , \text{id } \langle \text{DECLISTN} \rangle$

(4) $\langle \text{DECLISTN} \rangle \rightarrow \varepsilon$

(5) $\langle \text{STLIST} \rangle \rightarrow \text{s } \langle \text{STLISTN} \rangle$

(6) $\langle \text{STLISTN} \rangle \rightarrow ; \text{s } \langle \text{STLISTN} \rangle$

(7) $\langle \text{STLISTN} \rangle \rightarrow \varepsilon$

(8) $\langle \text{TYPE} \rangle \rightarrow \text{real}$

(9) $\langle \text{TYPE} \rangle \rightarrow \text{int}$

```
program DESCENT;
```

```
begin
```

```
    GETNEXT(TOKEN);
```

```
    PROGRAM(TOKEN);
```

```
    GETNEXT(TOKEN);
```

```
    if TOKEN ≠ '$' then ERROR;
```

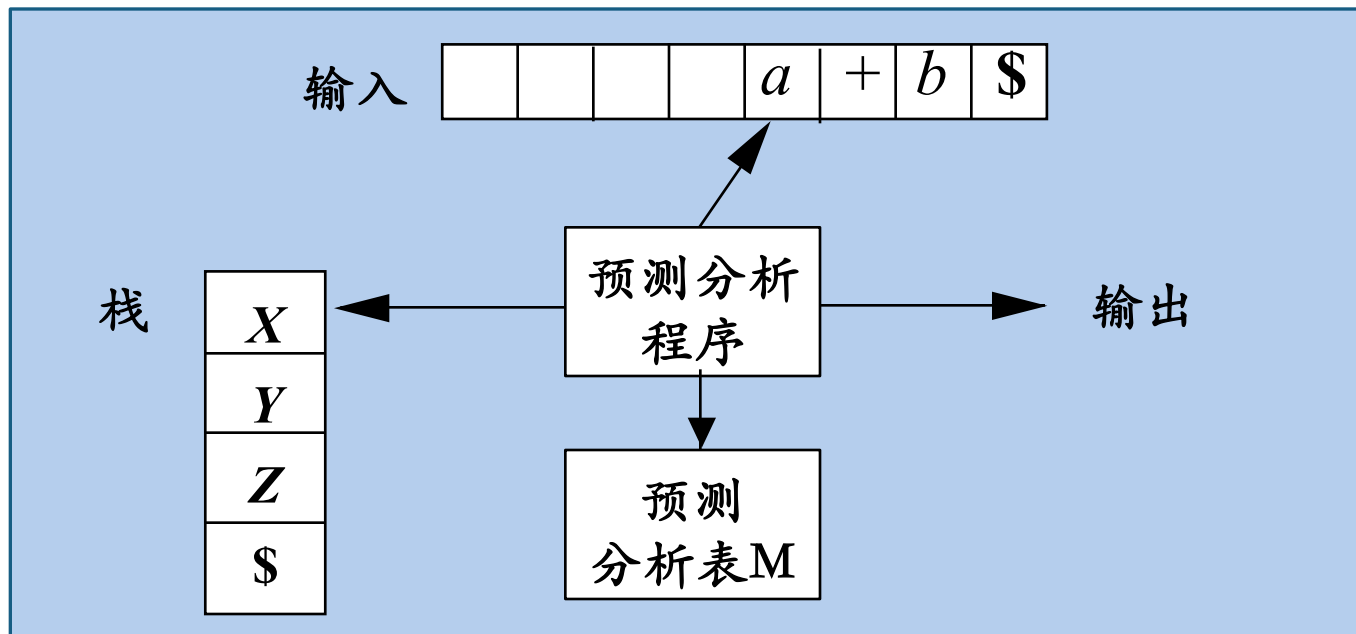
```
end
```

SELECT(4)={:}

SELECT(7)={end}

4.2.3 非递归的预测分析法

- 非递归的预测分析显式地维护一个栈结构，而不是通过递归调用的方式隐式地维护栈。这样的语法分析器可以模拟最左推导过程



例

最左推导

非终结符	输入符号					
	id	+	*	()	\$
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	$F \rightarrow \text{id}$			$F \rightarrow (E)$		

栈	剩余输入	输出
$E \$$	$\text{id}+\text{id}*\text{id} \$$	
$TE' \$$	$\text{id}+\text{id}*\text{id} \$$	$E \rightarrow TE'$
$FT'E' \$$	$\text{id}+\text{id}*\text{id} \$$	$T \rightarrow FT'$
$\text{id}T'E' \$$	$\text{id}+\text{id}*\text{id} \$$	$F \rightarrow \text{id}$
$T'E' \$$	$+\text{id}*\text{id} \$$	
$E' \$$	$+\text{id}*\text{id} \$$	$T' \rightarrow \varepsilon$
$+TE' \$$	$+\text{id}*\text{id} \$$	$E' \rightarrow +TE'$
$TE' \$$	$\text{id}*\text{id} \$$	
$FT'E' \$$	$\text{id}*\text{id} \$$	$T \rightarrow FT'$
$\text{id}T'E' \$$	$\text{id}*\text{id} \$$	$F \rightarrow \text{id}$
$T'E' \$$	$*\text{id} \$$	
$*FT'E' \$$	$*\text{id} \$$	$T' \rightarrow *FT'$
$FT'E' \$$	$\text{id} \$$	
$\text{id}T'E' \$$	$\text{id} \$$	$F \rightarrow \text{id}$
$T'E' \$$	$\$$	
$E' \$$	$\$$	$T' \rightarrow \varepsilon$
$\$$	$\$$	$E' \rightarrow \varepsilon$

表驱动的预测分析法

- 输入：一个串 w 和文法 G 的分析表 M
- 输出：如果 w 在 $L(G)$ 中，输出 w 的最左推导；否则给出错误指示
- 方法：最初，语法分析器的**格局**如下：输入缓冲区中是 $w\$$ ， G 的开始符号位于栈顶，其下面是 $\$$ 。下面的程序使用预测分析表 M 生成了处理这个输入的预测分析过程

```
设置 $ip$ 使它指向 $w$ 的第一个符号，其中 $ip$ 是输入指针；
令 $X$ =栈顶符号；
while (  $X \neq \$$  ) { /* 栈非空 */
    if (  $X$  等于  $ip$  所指向的符号  $a$  ) 执行栈的弹出操作，将 $ip$ 向前移动一个位置；
    else if (  $X$  是一个终结符号 )  $error()$ ；
    else if (  $M[X, a]$  是一个报错条目 )  $error()$ ；
    else if (  $M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$  ) {
        输出产生式  $X \rightarrow Y_1 Y_2 \dots Y_k$ ；
        弹出栈顶符号；
        将 $Y_k, Y_{k-1} \dots, Y_1$  压入栈中，其中 $Y_1$ 位于栈顶。
    }
    令 $X$ =栈顶符号
}
```

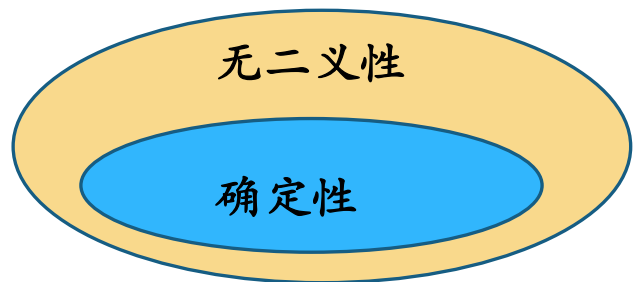

递归的预测分析法vs.非递归的预测分析法

	递归的预测分析法	非递归的预测分析法
程序规模	程序规模较大， 不需载入分析表	主控程序规模较小， 😊 需载入分析表（表较小）
直观性	较好 😊	较差
效率	较低	分析时间大约正比于待分 析程序的长度 😊
自动生成	较难	较易 😊

预测分析法实现步骤

- 1) 构造文法 无二义性
- 2) 改造文法: 消除二义性、消除左递归、消除回溯
- 3) 求每个变量的 $FIRST$ 集和 $FOLLOW$ 集, 从而求得每个候选式的 $SELECT$ 集 确定性
- 4) 检查是不是 $LL(1)$ 文法。若是, 构造预测分析表
- 5) 对于递归的预测分析, 根据预测分析表为每一个非终结符编写一个过程; 对于非递归的预测分析, 实现表驱动的预测分析算法

无二义性 vs. 确定性



二义性文法的判定

- 对于任意一个上下文无关文法，不存在一个算法，判定它是否为二义性的；但能给出一组充分条件，满足这组充分条件的文法是无二义性的
 - 满足，肯定无二义性
 - 不满足，也未必就是有二义性的

4.2.4 预测分析中的错误检测

- 两种情况下可以检测到错误
 - 栈顶的终结符和当前输入符号不匹配
 - 栈顶非终结符与当前输入符号在预测分析表对应项中的信息为空

非终结符	输入符号					
	id	+	*	()	\$
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	$F \rightarrow \text{id}$			$F \rightarrow (E)$		

预测分析中的错误恢复

➤ 恐慌模式 (Panic Mode)

- 忽略输入中的一些符号，直到输入中出现由设计者选定的 **同步词法单元** (*synchronizing token*) **集合** 中的某个词法单元
- 其效果依赖于 **同步集合的选取**。集合的选取应该使得语法分析器能从实际遇到的错误中 **快速恢复**
- 例如可以把 ***FOLLOW(A)*** 中的 **所有终结符** 放入非终结符 *A* 的同步记号集合
- 如果终结符在栈顶而不能匹配，一个简单的办法就是弹出此终结符

例

非终结符	输入符号					
	id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \text{id}$	synch	synch	$F \rightarrow (E)$	synch	synch

X	$\text{FOLLOW}(X)$
E	\$)
E'	\$)
T	+) \$
T'	+) \$
F	* +) \$

*Synch*表示根据相应非终结符的***FOLLOW***集得到的同步词法单元

➤ 分析表的使用方法

- 如果 $M[A,a]$ 是空，表示检测到错误，根据恐慌模式，忽略输入符号 a
- 如果 $M[A,a]$ 是*synch*，则弹出栈顶的非终结符 A ，试图继续分析后面的语法成分
- 如果栈顶的终结符和输入符号不匹配，则弹出栈顶的终结符



非终结符	输入符号					
	id	+	*	()	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \text{id}$	synch	synch	$F \rightarrow (E)$	synch	synch

栈	剩余输入	
$E \$$	$+id*+id \$$	ignore +
$E \$$	$id*+id \$$	
$TE' \$$	$id*+id \$$	
$FT'E' \$$	$id*+id \$$	
$idT'E' \$$	$id*+id \$$	
$T'E' \$$	$*+id \$$	
$*FT'E' \$$	$*+id \$$	
$FT'E' \$$	$+id \$$	error
$T'E' \$$	$+id \$$	
$E' \$$	$+id \$$	
$+TE' \$$	$+id \$$	
$TE' \$$	$id \$$	
$FT'E' \$$	$id \$$	
$idT'E' \$$	$id \$$	
$T'E' \$$	$\$$	
$E' \$$	$\$$	
$\$$	$\$$	



提纲

4.1 自顶向下的分析

4.2 预测分析法

4.3 自底向上的分析

4.4 LR分析法

4.5 语法分析器自动生成工具