



第四章 语法分析—— 自上而下分析

授课人：高珍

gaozhen@tongji.edu.cn

内容线索

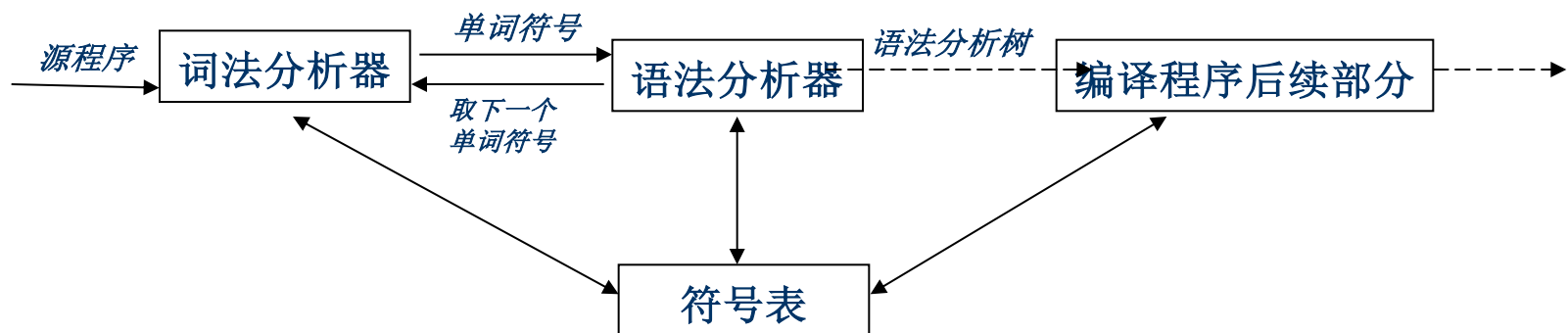
- 语法分析器的功能
- 自上而下分析方法概述
- LL (1) 分析方法
- 递归下降分析程序
- 预测分析程序

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(顶)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(底)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。

自顶向下分析例子

例1 文法G[Z]

$Z \rightarrow aBd$

$B \rightarrow d$

$B \rightarrow c$

$B \rightarrow bB$

求符号串abcd的推导过程

例2 文法G[S]

$S \rightarrow Ap | Bq$

$A \rightarrow a | cA$

$B \rightarrow b | dB$

求符号串ccap的推导过程

自底向上分析概述

■ 从终极字符串出发归约出文法的开始符

例1 文法G[Z]

$Z \rightarrow aBd$

$B \rightarrow d$

$B \rightarrow c$

$B \rightarrow bB$

求符号串abcd的归约过程

例2 文法G[S]

$S \rightarrow Ap|Bq$

$A \rightarrow a|cA$

$B \rightarrow b|dB$

求符号串ccap的归约过程

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

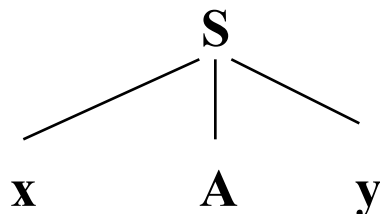
- 从文法的开始符号出发，向下推导，推出句子
- 对任何的输入串(单词符号)，试图用一切可能的办法, 从文法的开始符号出发，自上而下地为输入串建立一棵语法树，即为输入串寻找一个最左推导。

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

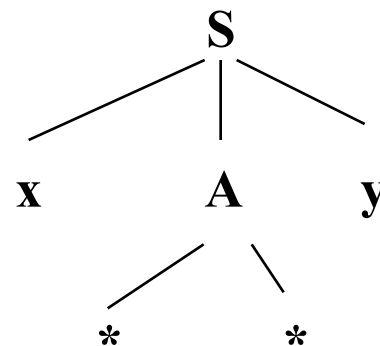
判定输入串 $x * y$ 是否为它的句子?

$x * y$
↑

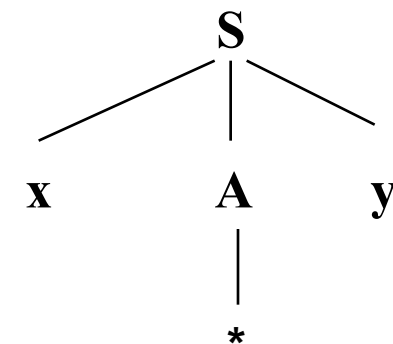
用 $S \rightarrow xAy$



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)

带回溯自上而下分析面临的问题

- 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

- 虚假匹配问题

- 回溯

- 回溯会引起时间和空间的大量消耗

- 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 $A \rightarrow aB$

$A \rightarrow Bb$

$B \rightarrow Ac$

$B \rightarrow d$

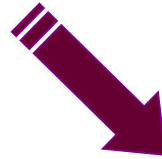
是间接左递归

消除直接左递归

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



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



β

$\beta\alpha$

$\beta\alpha\alpha$

$\beta\alpha\alpha\alpha$

... ..

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P关于的产生式是

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

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

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

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

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 $P_i \rightarrow P_j \gamma$ 的规则改写为 :

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

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

例.文法G(S) $R \rightarrow Sa | a$ $Q \rightarrow Rb | b$ $S \rightarrow Qc | c$

有推导： $S \Rightarrow Qc \Rightarrow Rbc \Rightarrow Sabc$, 存在左递归。

按R(1)、Q(2)、S(3)排序，执行算法得：

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

$i=2$ ， j 从1至1，R的产生式代入Q的产生式得： $Q \rightarrow Sab | ab | b$ ，无直接左递归。

$i=3$ ， j 从1至2:

$j=1$ ，S的候选式不含R，所以无需替换；

$j=2$ ，S的候选式含Q，将 $Q \rightarrow Sab | ab | b$ 代入S的候选式得：

$$S \rightarrow Sabc | abc | bc | c$$

再消除直接左递归得：

$$S \rightarrow abcS' | bcS' | cS'$$

$$S' \rightarrow abcS' | \varepsilon$$

消除无用产生式： $Q \rightarrow Sab | ab | b$ ， $R \rightarrow Sa | a$ ，

得文法： $S \rightarrow abcS' | bcS' | cS'$

$$S' \rightarrow abcS' | \varepsilon$$

文法对应的正规式： $V1 = (abc | bc | c)(abc)^*$ 。

例. 文法 $G(S)$ $S \rightarrow Qc \mid c$ $Q \rightarrow Rb \mid b$ $R \rightarrow Sa \mid a$

解： 1) 排序： $S(1)$ 、 $Q(2)$ 、 $R(3)$

2) 代入得： $S \rightarrow Qc \mid c$

$Q \rightarrow Rb \mid b$

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

消除直接左递归：

$S \rightarrow Qc \mid c$

$Q \rightarrow Rb \mid b$

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

$R' \rightarrow bcaR' \mid \varepsilon$

消除隐含的左递归算法与非终极符排序方法无关

随堂练习

■ 消去下面文法的左递归

$A \rightarrow aB$

$A \rightarrow Bb$

$B \rightarrow Ac$

$B \rightarrow d$

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。

示例无回溯！

回溯原因

若当前符号 = a ，下一步要展开 A ，而 $A \rightarrow \alpha_1 | \alpha_2 | \cdots | \alpha_n$ ，怎样选择 α_i ？

(1) 以 a 为头的 α_i 如果只有一个，则替换唯一；

(2) 若以 a 为头有多个 α_i 的，则替换不唯一，需要回溯，这是文法的问题，应该变换文法。

例子

文法： $S \rightarrow xAy$ $A \rightarrow ** | *$

句子： x^*y ； $x^{**}y$

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 γ_1 、 γ_2 、 \dots 、 γ_m 不以 δ 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1. 文法G: $S \rightarrow aSb | aS | \epsilon$

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

$S \rightarrow \epsilon$

引入新符: $S \rightarrow aSA$

$A \rightarrow b | \epsilon$

$S \rightarrow \epsilon$

例2. 文法G: $S \rightarrow abc | abd | ae$

解: 提取: $S \rightarrow a(bc | bd | e)$

引入新符: $S \rightarrow aA$

$A \rightarrow bc | bd | e$

引入新符 ...

无回溯

- 对任非终结符A，用它匹配输入串时能够根据当前输入，准确地指派一个候选式
 - 若匹配成功，则不虚假；
 - 若匹配不成功，则其它的候选式也不会成功。
- 即当A执行匹配时， $A \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$

若A面临的第一个输入符号为 a ，则应该准确地指派某一个 α_i ，其成败完全代表A，无需进行试探和回溯。

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 α 的**终结首符集** $\text{FIRST}(\alpha)$ 定义为:

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

特别地，若 $\alpha \Rightarrow^* \varepsilon$ ，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 $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \Phi, i \neq j$

计算FIRST(X)集

■ 对每一个文法符号X计算FIRST(X)

- 若 $X \in V_T$, $\text{FIRST}(X) = \{X\}$
- 若 $X \in V_N$, $\text{FIRST}(X) = \{a \mid X \rightarrow a \cdots, a \in V_T\}$
- 若 $X \in V_N$, 且有产生式 $X \rightarrow \epsilon$, 则 $\{\epsilon\} \in \text{FIRST}(X)$
- 若 $X \in V_N$, 且有产生式 $X \rightarrow Y_1 Y_2 \dots Y_n$, 且 $Y_1 Y_2 \dots Y_n \in V_N$
 - 当 $Y_1, Y_2, \dots, Y_{i-1} \xRightarrow{*} \epsilon$, 则 $\text{FIRST}(Y_1) - \{\epsilon\}, \text{FIRST}(Y_2) - \{\epsilon\} \dots \text{FIRST}(Y_{i-1}) - \{\epsilon\}, \text{FIRST}(Y_i)$ 都包含在 $\text{FIRST}(X)$ 中
 - 当 $Y_i \xRightarrow{*} \epsilon (i=1, 2 \dots n)$, 将 $\{\epsilon\}$ 并入 $\text{FIRST}(X)$ 中

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

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

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

随堂练习(Canvas)

文法规则	FIRST(X)
$E \rightarrow E A T \mid T$	
$A \rightarrow + \mid -$	
$T \rightarrow T M F \mid F$	
$M \rightarrow *$	
$F \rightarrow (E) \mid n$	

FIRST (α) 构造

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

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

该算法有什么问题吗？

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

随堂练习

■ 文法G[S]

$$\square S \rightarrow aA \mid d$$

$$\square A \rightarrow bS \mid \varepsilon$$

■ 对于输入串abd，根据FIRST (α) 方法来求其自顶向下的推导过程

[提示 FIRST(aA)= a]

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 $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \Phi$
- 对某个输入符号 a ，及待匹配的非终结符 A
($A \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$)， a 不属于任何候选式的 FIRST 集合，即对任意 α_i ， $a \notin \text{FIRST}(\alpha_i)$
- 此时，该如何选择 A 的候选式，或者就认为 a 的出现是一种语法错误？

示例

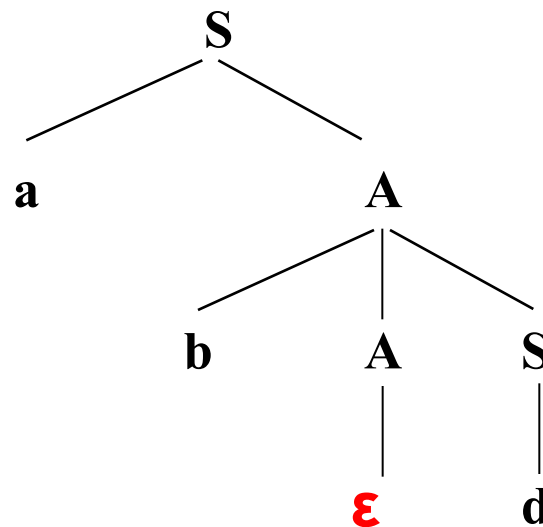
$$\text{FIRST}(S) = \{a, b\}$$

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

例. $G(S): S \rightarrow aA \mid d$
 $A \rightarrow bAS \mid \epsilon$

输入符号串abd是否为句子？

$$\begin{aligned} S &\Rightarrow aA \\ &\Rightarrow abAS \\ &\Rightarrow abS \\ &\Rightarrow abd \end{aligned}$$



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集合

- 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：

$$\text{FOLLOW}(A) = \{ a \mid S \xRightarrow{*} \cdots Aa \cdots, a \in V_T \}$$

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

$$\# \in \text{FOLLOW}(A)。$$

FOLLOW(A)是所有句型中出现在紧接A之后的终结符或“#”。

LL(1)文法条件完善

- 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 ϵ （即 $\epsilon \in \text{FIRST}(A)$ ），那么，当 $a \in \text{FOLLOW}(A)$ 时，就允许A自动匹配（即选用 $A \rightarrow \epsilon$ 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 ϵ ，则

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

课堂测试(Canvas)

- LL(1)条件

FOLLOW(A)的构造

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 $\text{FIRST}(\beta) - \{\epsilon\}$ 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: $\text{FOLLOW}(E) = \{ \#,) \}$

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

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

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

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

随堂练习(Canvas)

文法规则	FOLLOW(X)
$E \rightarrow EAT T$	
$A \rightarrow + -$	
$T \rightarrow TMF F$	
$M \rightarrow *$	
$F \rightarrow (E) n$	

随堂练习

■ 文法G[S]

$$\square S \rightarrow aA \mid d$$

$$\square A \rightarrow bAS \mid \varepsilon$$

- 对于输入串abd，根据**FIRST**
(α)/FOLLOW(A)方法来求其自顶向下的推
导过程

[提示 $\text{FIRST}(aA)=a$]

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

文法每一个非终结符A的任何两个不同产生式 $A \rightarrow \alpha_i \mid \alpha_j$, 有 $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \emptyset$

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

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

(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 ϵ ，则 $\text{FIRST}(A) \cap \text{FOLLOW}(A) = \emptyset$,

则称该文法G为**LL(1)文法**。

文法每一个非终结符A，若 $A \Rightarrow \epsilon$ 则 $\text{FIRST}(A) \cap \text{FOLLOW}(A) = \emptyset$

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 α ，要用非终结符 A 进行匹配，且 $A \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$ ，则可如下分析：
 - (1) 若 $\alpha \in \text{FIRST}(\alpha_i)$ ，则指派 α_i 执行匹配任务；
 - (2) 否则
 - 1) 若 $\epsilon \in \text{FIRST}(A)$ ，且 $\alpha \in \text{FOLLOW}(A)$ ，则让 A 与 ϵ 自动匹配；
 - 2) 否则， α 的出现是一种语法错误。

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。

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;

// else? $ch \in$
 $\text{FOLLOW}(T')$?

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;



完善优化上述代码

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

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

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

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

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

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

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

```
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;
```

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 $\alpha=y_1y_2y_3\cdots y_m$, 相应的子程序 $P(\alpha)$ 为:

BEGIN $P(y_1)$

$P(y_2)$

 ...

$P(y_m)$

END

- 如果 $y_i \in V_T$, 则 $P(y_i)$ 为:

IF $ch = y_i$ 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

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

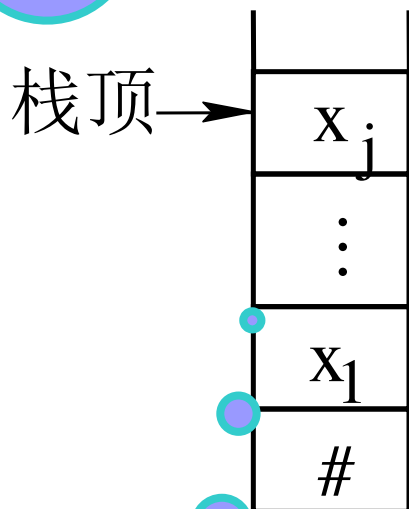
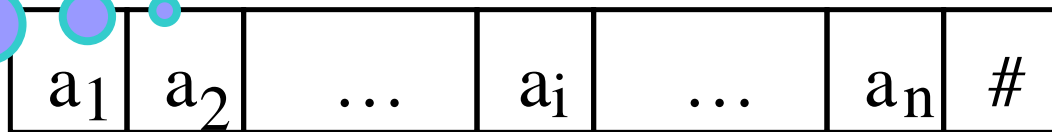
- 使用一个分析表和符号栈进行联合控制，是实现LL(1)分析的另一种有效方法。

预测分析程序基本思想

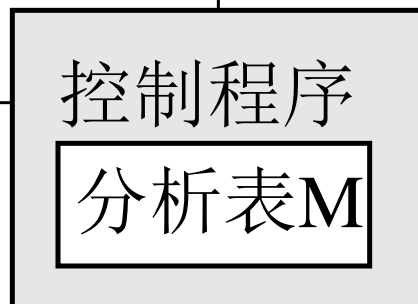
- 根据输入串的当前输入符确定选用某一个产生式进行推导，当该输入符与推导的第一个符号相同时，再取输入串的下一个符号，继续确定下一个推导应选的产生式，如此下去，直到推出被分析的输入串为止。
- 预测分析程序（LL(1)）分析器组成
 - LL(1)分析表（预测分析表）
 - 符号栈（后进先出）
 - 控制程序（表驱动程序）组成。

待分析的符号串,它以“#”作为结束标志。

串:



分析栈



输出

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 $\#$
 - $M[A, \alpha]$ 为一条关于 A 的产生式，指出当 A 面临 α 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	$F \rightarrow i$			$F \rightarrow (E)$		

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 #

↑

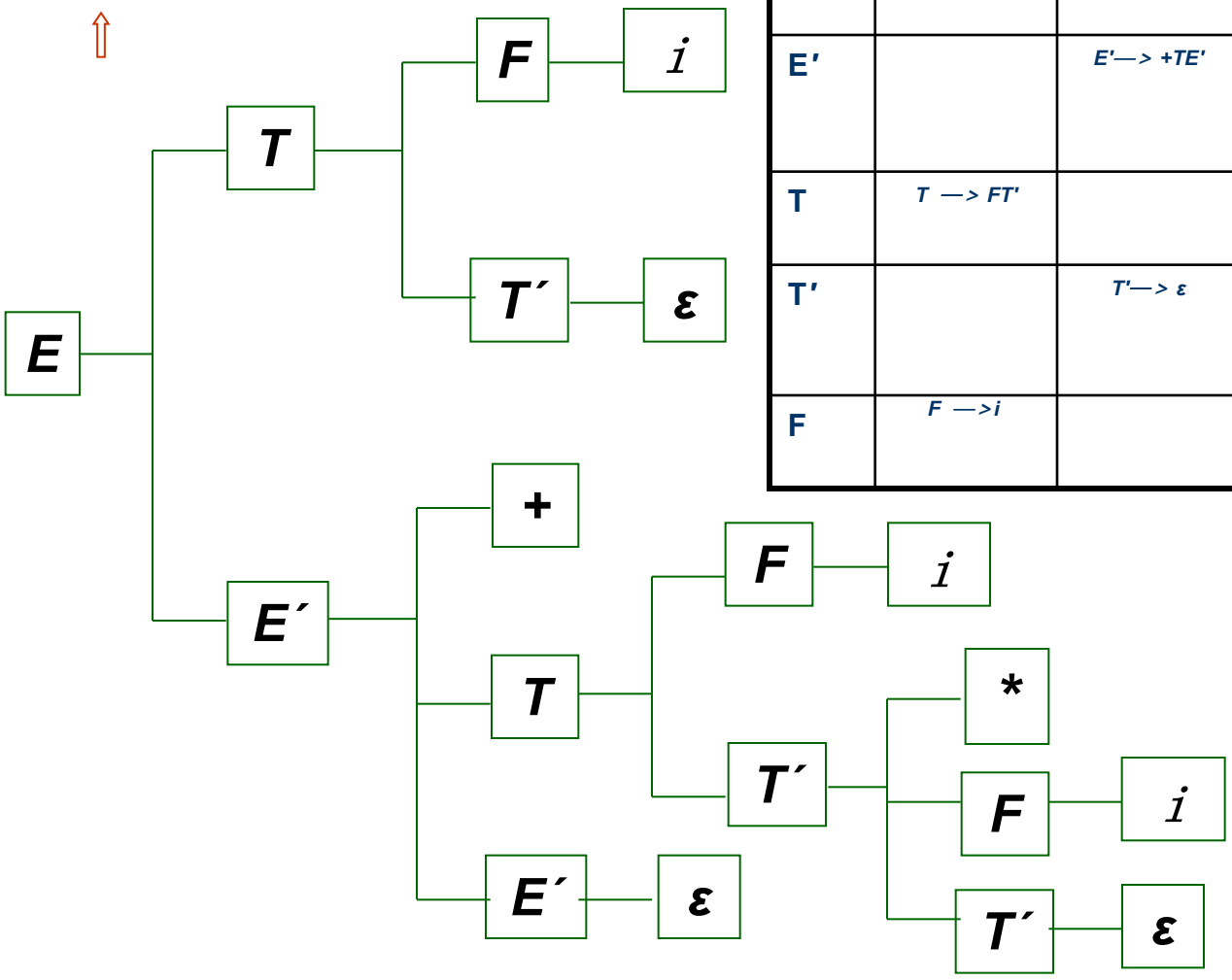
P80

	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)		

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

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

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

i + i * i #

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

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： $A \xRightarrow{*} \varepsilon$

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 $\varepsilon \in \text{FIRST}(A)$ ，对 $b \in \text{FOLLOW}(A)$ 置 $M[A, b] = A \rightarrow \varepsilon$
- (4) 其余置 $M[A, a] = \text{ERROR}$

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

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

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

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

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

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

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

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

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

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

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) $FIRST(\alpha) \cap FIRST(\beta) = \Phi$
 - (2) 若 $\beta \Rightarrow^* \varepsilon$ ，则 $FIRST(A) \cap FOLLOW(A) = \Phi$

总结

- 消除左递归算法
- 提取公共左因子法
- $\text{FIRST}(\alpha)$ / $\text{FIRST}(X)$ 构造算法
- $\text{FOLLOW}(X)$ 构造算法
- 递归下降分析程序构造算法
- LL(1)分析表生成算法
- 预测分析程序构造算法

Canvas作业

- 作业4-LL(1)文法
 - P81 (1)

Dank u

Dutch

Merci

French

Спасибо

Russian

Gracias

Spanish

شكراً

Arabic

धन्यवाद

Hindi

감사합니다

Korean

תודה רבה

Hebrew

Tack så mycket

Swedish

Obrigado

Brazilian
Portuguese

Dankon

Esperanto

Thank You !

谢谢

Chinese

ありがとうございます

Japanese

Trugarez

Breton

Danke

German

Tak

Danish

Grazie

Italian

நன்றி

Tamil

děkuji

Czech

ขอบคุณ

Thai

go raibh maith agat

Gaelic