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

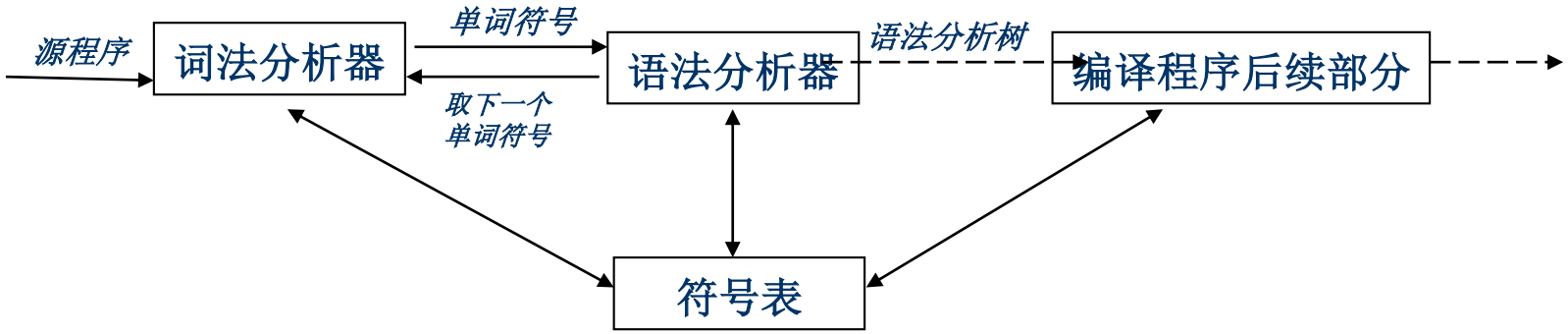
- **语法分析器的功能**
- **自上而下分析方法概述**
- **LL（1）分析方法**
- **递归下降分析程序**
- **预测分析程序**

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
- 自下（底）而上分析
  - 算符优先分析法
  - LR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。

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 → aBd

B → d

B → c

B → bB

求符号串abcd的推导过程

例2 文法G[S]

S → Ap|Bq

A → a|cA

B → b|dB

求符号串ccap的推导过程

自底向上分析概述

- 从终极符串出发归约出文法的开始符

例1 文法G[Z]

Z → aBd

B → d

B → c

B → bB

求符号串abcd的归约过程

例2 文法G[S]

S → Ap|Bq

A → a|cA

B → 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。

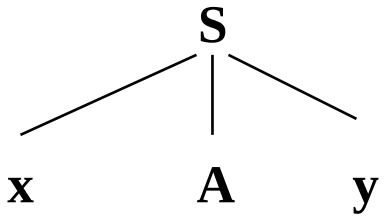
例. 设文法G[S]： S→xAy，A→**|*

判定输入串 x * y是否为它的句子?

x * y

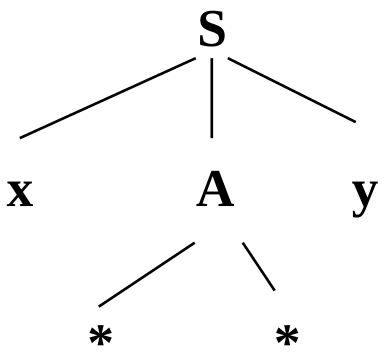
↑

用S→xAy



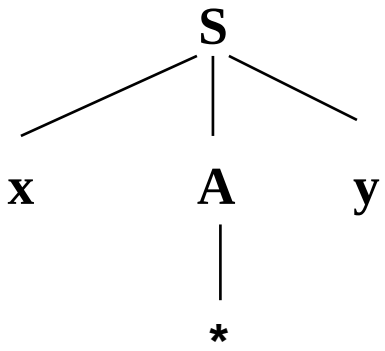
用A→**

(回溯)



用A→*

(成功)



推导过程:

S ⇒ xAy

⇒ x**y(回溯)

⇒ 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

$$P \overset{+}{\Rightarrow} P\alpha$$

  - 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(Left)到右扫描输入串；构造最左(Leftmost)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

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

  b)间接递归

  $A \rightarrow B\beta$

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

  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 →Sa

S →b

是直接左递归

语言是：$L = \{ ba^n \mid n \geq 0 \}$

例2. 文法 A→aB

A→Bb

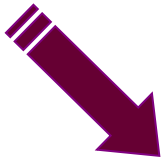
B→Ac

B→d

是间接左递归

消除直接左递归

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



$P \rightarrow \beta P'$

$P' \rightarrow \alpha P' \mid \varepsilon$

β

$\beta\alpha$

$\beta\alpha\alpha$

$\beta\alpha\alpha\alpha$

…… ……

例. 文法 $E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid i$

$E \to TE'$

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

$T \to FT'$

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

$F \to (E) \mid i$

- **一般地, 假定P关于的产生式是**

  $P \to P\alpha_1 \mid P\alpha_2 \mid \ldots \mid P\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

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

  则改写为： $P \to \beta_1 P' \mid \beta_2 P' \mid \ldots \mid \beta_n P'$

  $P' \to \alpha_1 P' \mid \alpha_2 P' \mid \ldots \mid \alpha_m P' \mid \varepsilon$

随堂练习

- **消去下面文法的左递归**

  $T \rightarrow T,S|S$

  $S \rightarrow a|+|(T)$

消除左递归算法

(1) 排序: P_1、P_2、…、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:

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

其中: $P_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ 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→Sa|a　　Q→Rb|b　　S →Qc|c

有推导：S ⇒ Qc ⇒ Rbc ⇒ 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→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→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→Sab|ab|b代入S的候选式得：

S→Sabc|abc|bc|c

再消除直接左递归得：

S→abcS'| bcS'| cS'

S'→abcS'|ε

消除无用产生式：Q→Sab|ab|b，R→Sa|a，

得文法：　　S→abcS'| bcS'| cS'

S'→abcS'|ε

文法对应的正规式：V1=(abc|bc|c)(abc)* 。

例. 文法G(S) S →Qc|c　　Q→Rb|b　　R→Sa|a

解：　1）排序：S(1)、Q(2)、R(3)

2）代入得：　S →Q c | c

Q →R b | b

R →Rbca | bca | ca | a

消除直接左递归：

S →Q c | c

Q →R b | b

R →bcaR ' | caR' | aR '

R' →bcaR' | ε

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→aB

  A→Bb

  B→Ac

  B→d

回溯问题

例如，有产生式：

语句→ 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！

回溯原因

若当前符号 = a，下一步要展开A ，而 $A \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$ ，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

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， $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

  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FIRST (α)** 定义为：

  $FIRST(\alpha) = \{ a \mid \alpha \stackrel{*}{\Rightarrow} a\ldots , a \in V_T \}$

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

- 以上条件（2）可表示为：对文法的任一非终结符号A，若

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

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

计算FIRST(X)集

- **对每一个文法符号X计算FIRST(X)**
  - 若$X \in V_T$, FIRST(X)={X}
  - 若$X \in V_N$, $FIRST(X)=\{a \mid X \rightarrow a\ldots, a \in V_T\}$
  - 若$X \in V_N$, 且有产生式$X \rightarrow \varepsilon$，则$\{\varepsilon\} \in FIRST(X)$
  - 若$X \in V_N$, 且有产生式$X \rightarrow Y_1 Y_2 \ldots Y_n$, 且$Y_1 Y_2 \ldots Y_n \in V_N$
    - 当Y_1，Y_2，…，$Y_{i-1} \Rightarrow \varepsilon$，则$FIRST(Y_1)-\{\varepsilon\}$, $FIRST(Y_2)-\{\varepsilon\}$… $FIRST(Y_{i-1})-\{\varepsilon\}$, $FIRST(Y_i)$都包含在FIRST(X)中
    - 当$Y_i \Rightarrow \varepsilon (i=1,2\ldots n)$,将$\{\varepsilon\}$并入FIRST(X)中

*

*

例. G: E → TE'

E' → + TE' | ε

T →FT'

T' → *FT' | ε

F →(E) | i

求每个非终结符号的FIRST集合

解：FIRST (E) = FIRST (T)

= FIRST (F)

= { (, i }

FIRST (E') = { +, ε }

FIRST (T') = { *, ε}

随堂练习

文法规则	FIRST(X)
E → E A T | T	
A → + | -	
T → T M F | F	
M → *	
F → (E) | n	

FIRST（α）构造

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

(1) 置 FIRST(α) = FIRST (X_1) - {ε};

(2) 若对所有的 X_j ,1<=j<= i -1, ε∈FIRST (X_j), 则把FIRST(X_j) -{ε}加到FIRST(α)中;

……

(3) 若对所有的 X_j ,1<= j <=n, ε∈FIRST (X_j), 则把ε加到FIRST(α)中。

补充完整

例. G: E → TE'

E' → + TE' | ε

T →FT'

T' → *FT' | ε

F →(E) | i

求每个产生式右部符号串的FIRST集合

解：FIRST(TE') = { (, i }

FIRST(+TE') = { + }

FIRST(FT') = { (, i }

FIRST(*FT') = { * }

FIRST ((E)) = { (}

FIRST (i) = { i }

随堂练习

- 文法G[S]
  - S → aA|d
  - A → bS| ε
- 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]

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。**

$$A \to \delta\beta_1 \mid \delta\beta_2 \mid \cdots \mid \delta\beta_n \mid \gamma_1 \mid \gamma_2 \mid \cdots \mid \gamma_m$$

(其中γ_1、 γ_2、 …、 γ_m不以δ开头)

$$A \to \delta A' \mid \gamma_1 \mid \gamma_2 \mid \cdots \mid \gamma_m$$

$$A' \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

例. 文法G：S→aSb|aS|ε

解：提取：　S →aS(b|ε)

S→ε

引入新符：　S →aSA

A→ b|ε

S→ε

例. 文法G：S→abc|abd|ae

解：提取：　S →a(bc|bd|e)

引入新符：　S →aA

A→ bc|bd|e

引入新符 …

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$FIRST(\alpha_i) \cap FIRST(\alpha_j) = \Phi$
- 对某个输入符号a，及待匹配的非终结符A（$A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$），a不属于任何候选式的FIRST集合，即对任意$\alpha_i$，$a \notin FIRST(\alpha_i)$
- 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

FIRST(S)={a, b}

FIRST(A) ={b, ε}

例. G(S):S → aA|d

A → bAS|ε

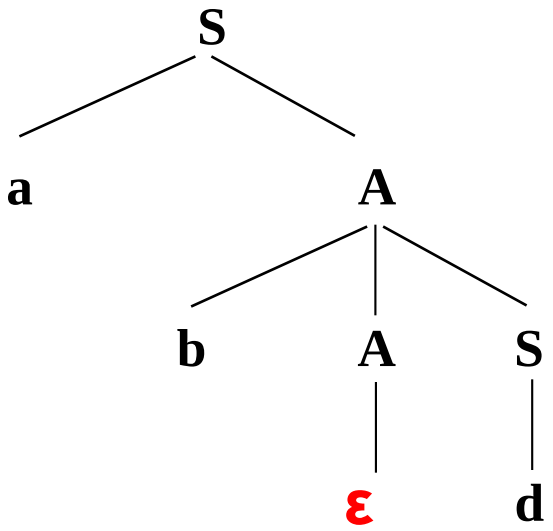
输入符号串abd是否为句子?

S ⇒ aA

⇒ abAS

⇒ abS

⇒ abd



这是因为A有产生式A→ε,而从开始符号S可以得出

$S \overset{*}{\Rightarrow} \ldots Ad \ldots$

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：

  $FOLLOW(A) = \{ a \mid S \overset{*}{\Rightarrow} \ldots Aa\ldots , a \in V_T \}$

- 特别地,若$S \overset{*}{\Rightarrow} \ldots A$，则规定

  $\# \in FOLLOW(A)$。

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$a \notin FIRST(\alpha_i)$ (对任意i)时，如果A的某个候选首符集包含ε（即ε∈FIRST（A）），那么，当a∈FOLLOW（A）时，就允许A自动匹配（即选用A→ε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

$$FIRST(A) \cap FOLLOW(A) = \Phi$$

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→αAβ,

则将FIRST(β) - {ε}加到FOLLOW (A)中;

(3) 若有B→αA或B →αAβ, 且β $\stackrel{*}{\Rightarrow}$ ε

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 → TE'

E' → + TE' | ε

T →FT'

T' → *FT' | ε

F →(E) | i

求每个非终结符号的FOLLOW集合

解: FOLLOW (E) = { #,)}

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

FOLLOW (T) = { +, #,) }

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

FOLLOW (F) = { *, +, #,) }

随堂练习

文法规则	FOLLOW()
E → E AT	
E → T	
A→ +	
A→ -	
T → T M F	
T → F	
M → *	
F → (E)	
F → n	

随堂练习

- 文法G[S]
  - S → aA|d
  - A → bAS| ε
- 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 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

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

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

（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ε，则$FIRST(A) \cap FOLLOW(A) = \Phi$，

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 $A\rightarrow\alpha_1|\alpha_2|\dots|\alpha_n$，则可如下分析:

  (1) 若$a\in FIRST(\alpha_i)$，则指派α_i执行匹配任务；

  (2) 否则

  1) 若$\varepsilon\in FIRST(A)$，且$a\in FOLLOW(A)$，则让A与ε自动匹配；

  2) 否则，a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→TE'; E'→+TE' | ε ; T→FT' ; T ' →*FT ' |ε ; F→(E)|i

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

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

FIRST((E))={ (}　　FIRST(i)={ i }

```
E→TE'

P(E);
Begin
    P(T); P( E' );
End;
```

```
E' →+TE' | ε
P( E' );
Begin
   If ch = '+' Then
   begin
      read(ch);
      P( T ); P( E' );
    End;
   // ch ∈ FOLLOW(E' )?
End;
```

例. E→TE'; E'→+TE' | ε ; T→FT' ; T' →*FT' |ε ; F→(E)|i

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

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

FIRST((E))={ (}　　FIRST(i)={ i }

```
T→FT'


P( T );
Begin
   P( F );  P( T' );
End;
```

```
T' →*FT' | ε
P( T' );
Begin
    If ch='*' Then
    begin
        read(ch);
        P( F);  P( T' );
     End;
      // else? ch ∈
      FOLLOW(T')?
End;
```

例. E→TE'; E'→+TE' | ε ; T→FT' ; T' →*FT' |ε ; F→(E)|i

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

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

FIRST((E))={ (}　　FIRST(i)={ i }

```
F→(E)|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→TE'; E'→+TE' | ε ; T→FT' ; T' →*FT' |ε ; F→(E)|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( 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;
```

```
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;
```

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$A \rightarrow \alpha_1 | \alpha_2 | \cdots | \alpha_n$ 相应的子程序P(A)构造如下：

```
IF ch IN FIRST(α1) THEN P(α1)
ELSE IF ch IN FIRST(α2) THEN P(α2)
ELSE ……
ELSE IF ch IN FIRST(αn) THEN P(αn)
ELSE IF (ε∈FIRST(A) )AND (ch  IN FOLLOW(A) )
          THEN RETURN
ELSE ERROR
```

(3)对于符号串$\alpha=\gamma_1\gamma_2\gamma_3\cdots\gamma_m$，相应的子程序P(α)为：

BEGIN　P (γ_1)

P (γ_2)

…

P (γ_m)

END

- 如果$\gamma_i \in V_T$，则P（γ_i）为:

  IF ch= γ_i THEN read(ch)

  ELSE ERROR;

- 如果$\gamma_i \in V_N$，则P（γ_i）为上述（2）中相应的子程序

课堂练习

- P81 2(4)

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

同济的清晨

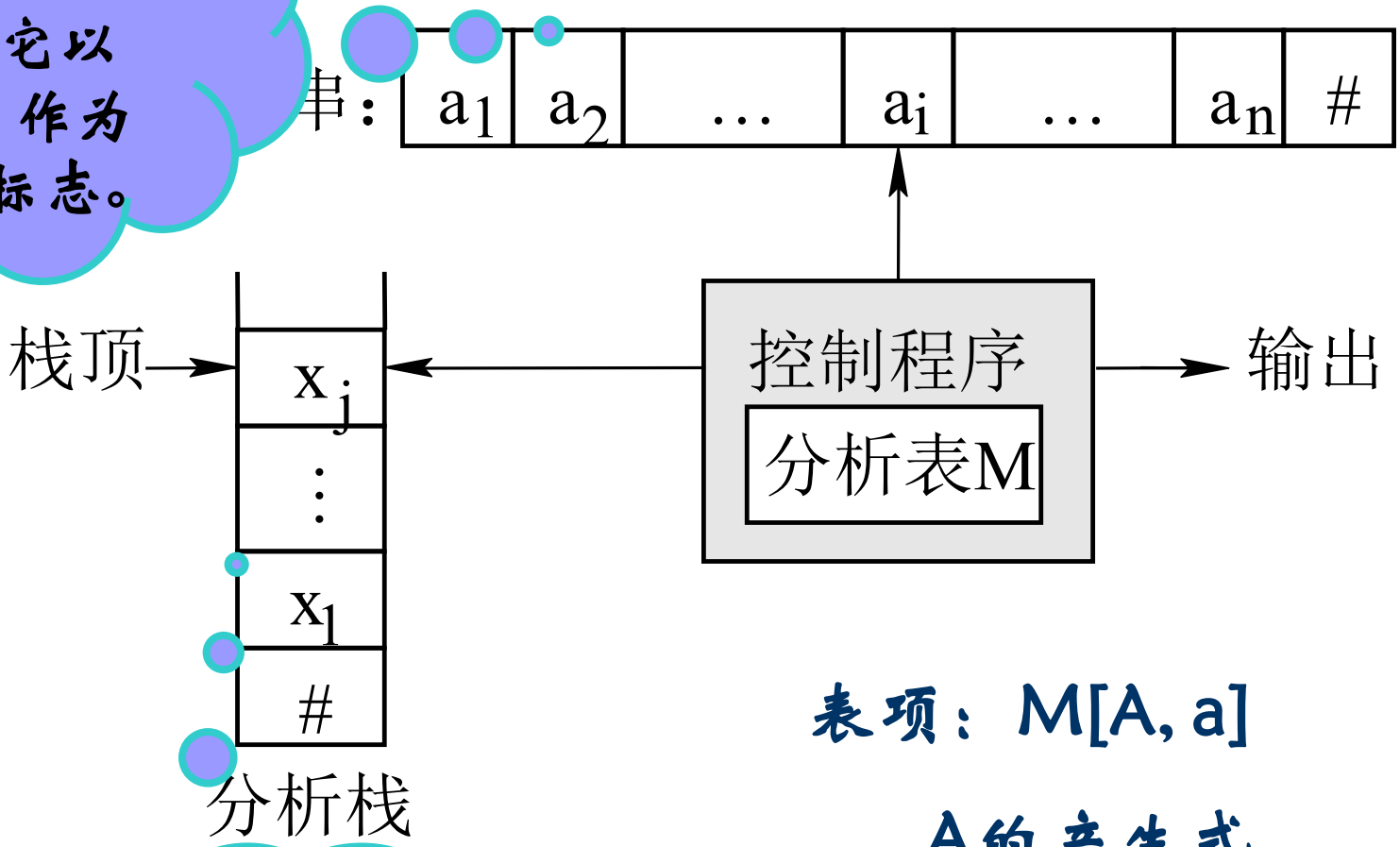
预测分析程序

- 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→TE'; E'→+TE' | ε ;

T→FT' ; T' →*FT' |ε ;

F→(E)|i

LL(1)分析表

	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

- 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 = “#”，即STACK 栈顶符号为“#”，当前输入符号为“#”，则分析成功。

(2) 若x = a≠“#”，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→ε，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=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 = " # " THEN

IF X = a THEN FLAG := FALSE ELSE ERROR

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

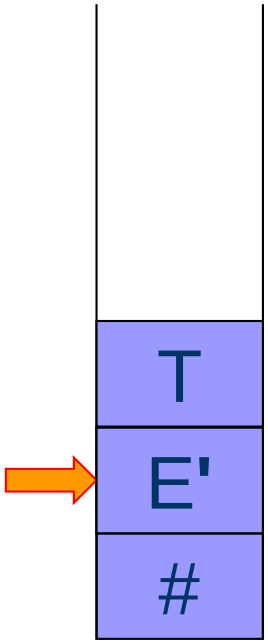
THEN 把$X_k, X_{k-1}, \ldots, X_1$ 逐一进栈

ELSE ERROR

END OF WHILE;

END

i + i * i #

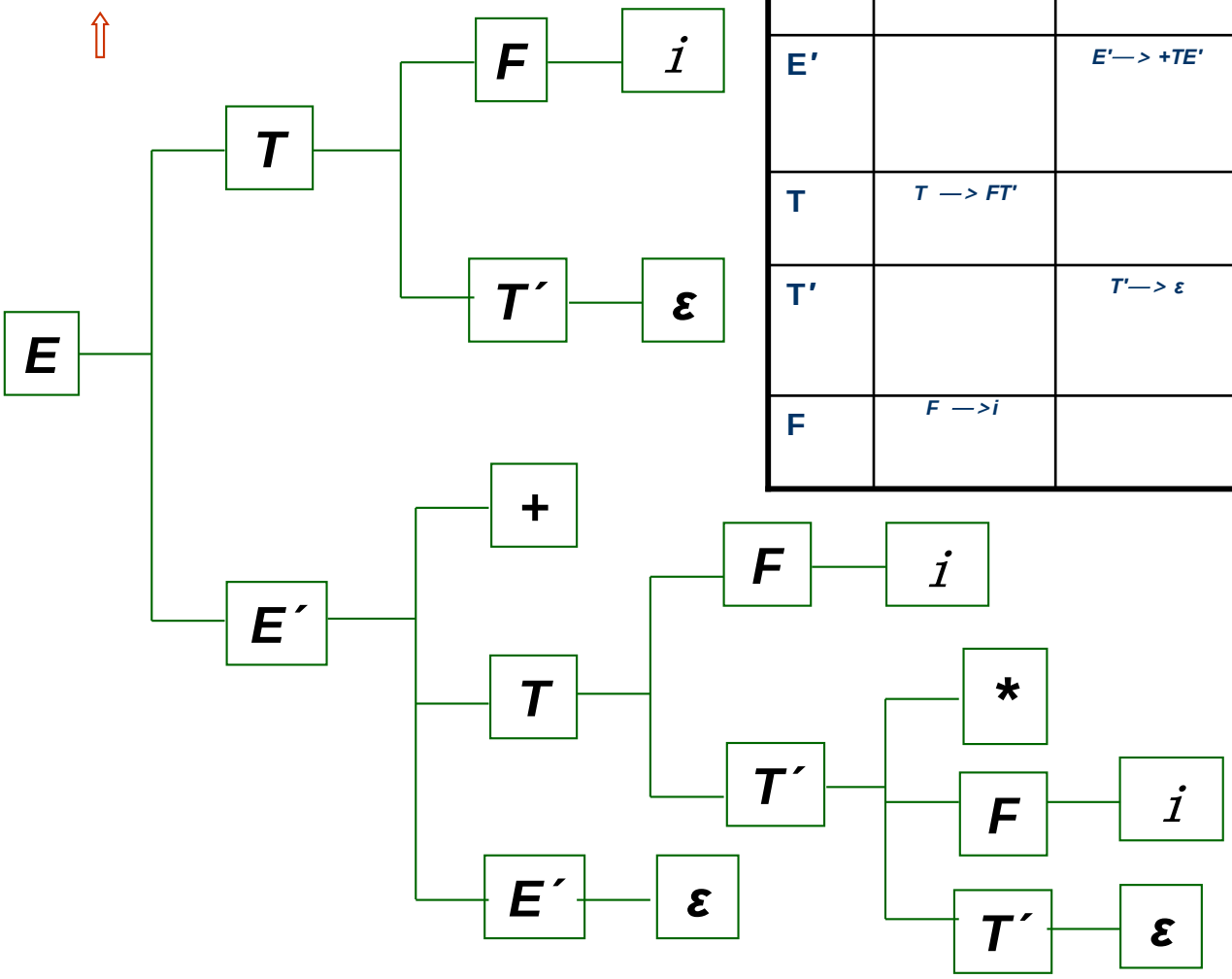


P80

	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 —>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)		

例. E→TE'; E'→+TE' | ε ;
T→FT' ; T' →*FT' |ε ;
F→(E)|i

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

i + i * i #

	栈	输 入	产生式
0	#E	i+i*i#	
1	#E′T	i+i*i#	E → TE′
2	#E′T′F	i+i*i#	T → FT′
3	#E′T′i	i+i*i#	F → i
4	#E′T′	+i*i#	
5	#E′	+i*i#	T′ → ε
6	#E′T+	+i*i#	E′ → +TE′
7	#E′T	i*i#	
8	#E′T′F	i*i#	T → FT′
9	#E′T′i	i*i#	F → i
10	#E′T′	*i#	
11	#E′T′F*	*i#	T′ → *FT′
12	#E′T′F	i#	
13	#E′T′i	i#	F → i
14	#E′T′	#	
15	#E′	#	T′ → ε
16	#	#	E′ → ε

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 $\overset{*}{\Rightarrow}$ ？ 将产生式分为两种：

  一种是：A $\overset{*}{\Rightarrow}$ a…

  另种是：A $\overset{*}{\Rightarrow}$ ε

分析表构造算法

(1) 对每个产生式 A→α，执行（2） 和（3）

(2) 若 a∈FIRST(α)， 置M[A, a] =A→α

(3) 若ε∈FIRST(A)，对b∈FOLLOW(A) 置 M[A, b] =A→ε

(4) 其余置 M[A, a] =ERROR

例. E→TE'; E'→+TE' | ε ; T→FT' ; T' →*FT' |ε ; F→(E)|i

FIRST(TE')={ (, i }　　FIRST(+ TE')={ + }

FIRST(FT')={ (, i }　　FIRST(*FT ')={ * }

FIRST((E))={ (}　　FIRST(i)={ i }

FOLLOW(E ')={),# }　　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 -> α| β,有：

（1）FIRST(α)∩ FIRST(β) = Φ

（2）若 β⇒* ε ，则 FIRST(A)∩ FOLLOW(A) =Φ

Dank u
Dutch

Merci
French

Спасибо
Russian

Gracias
Spanish

شكراً
Arabic

감사합니다
Korean

Tack så mycket
Swedish

धन्यवाद
Hindi

תודה רבה
Hebrew

Obrigado
Brazilian Portuguese

Thank You !

谢谢
Chinese

Dankon
Esperanto

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