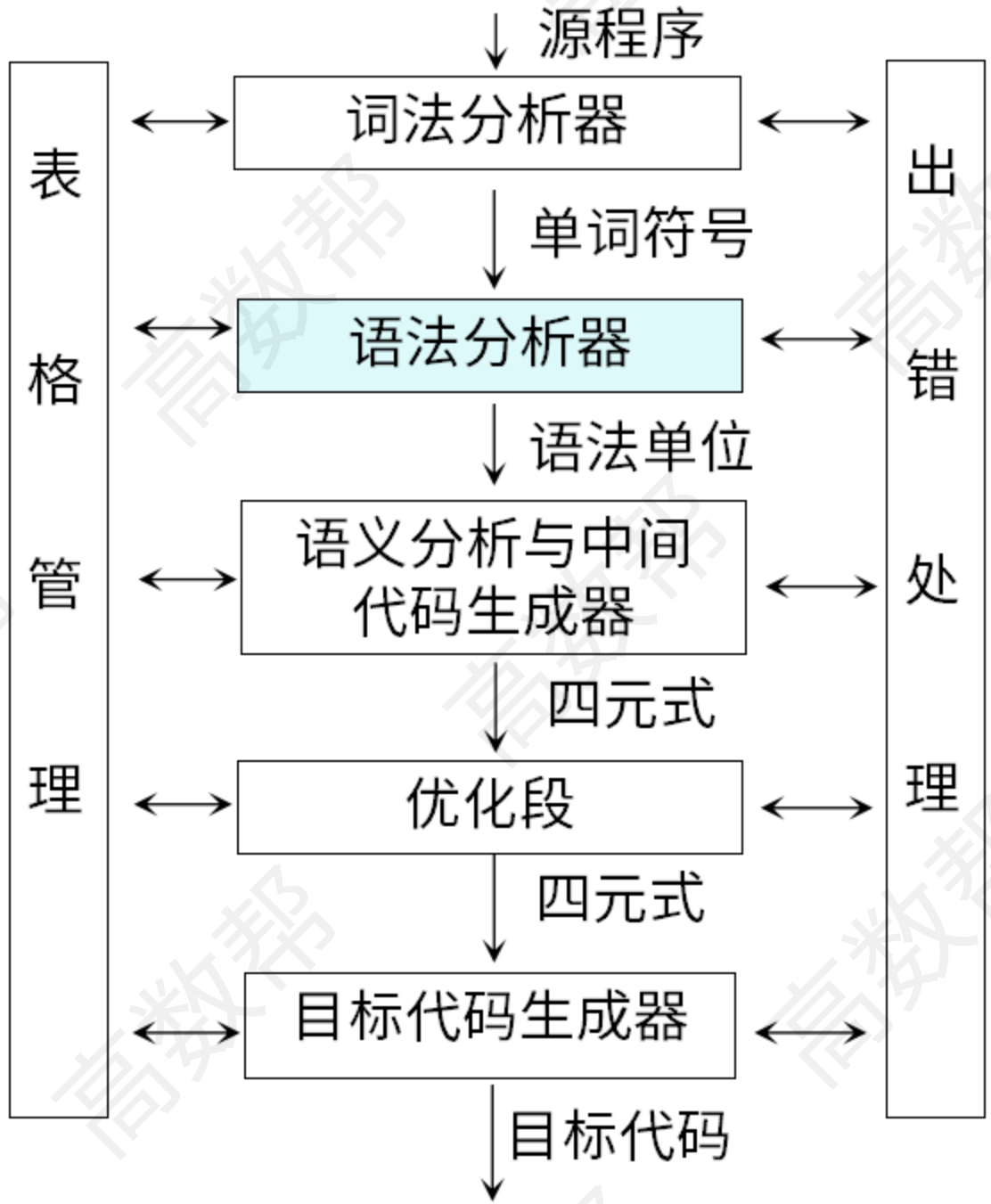
课时4 语法分析-自上而下

高数帮

考点	重要程度	占分	题型
1. 语法分析相关概念	★★★	5~10	填空题、简答题
2. LL(1)文法的判定	★★★★	5~10	问答题
3. 消除左递归	★★★★★	5~10	简答题
4.FIRST & FOLLOW集的计算	★★★★★	5~10	问答题
5. 预测分析总控程序	★★★★	5~10	填空题
6. LL(1)分析法的过程	★★★★★	10~20	问答题
7. 递归下降子程序法	★★★	5~10	简答题


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间
代码生成器
四元式
优化段
四元式
目标代码生成器
目标代码
表
格
管
理
出
错
处
理

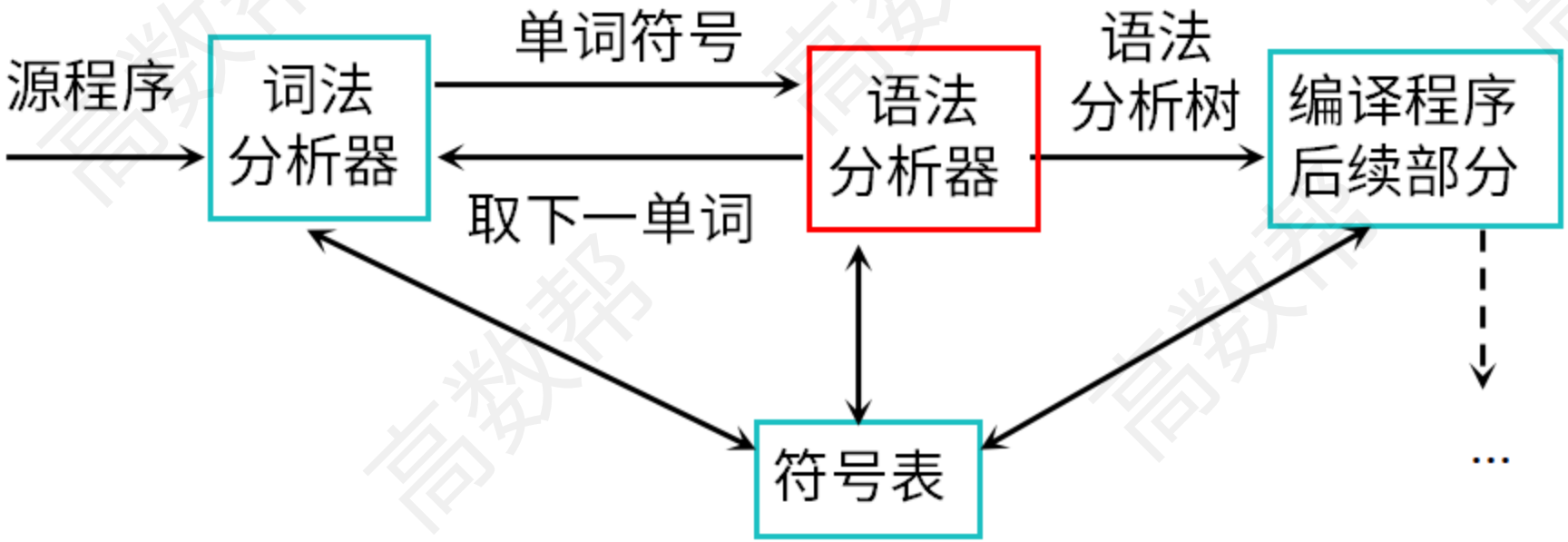
4.1 语法分析相关概念

一、语法分析器的功能

1.语法分析的任务：分析一个文法的句子结构。

2.语法分析器的功能：按照文法的产生式(语言的语法规则)，识别输入符号串是否为一个句子(合式程序)。

注：语法分析是编译程序的核心部分



二、语法分析的方法

1.自下而上分析法

基本思想：从输入串开始，逐步进行“**归约**”，直到文法的开始符号。即从树末端开始，构造语法树。所谓归约，是指根据文法的产生式规则，把产生式的右部替换成左部符号。

算符优先分析法：按照算符的优先关系和结合性质进行语法分析。适合分析表达式。

LR分析法：规范归约

【题1】 G(E)： $E \rightarrow i| E+E | E-E | E*E | E/E | (E)$

i*i+i

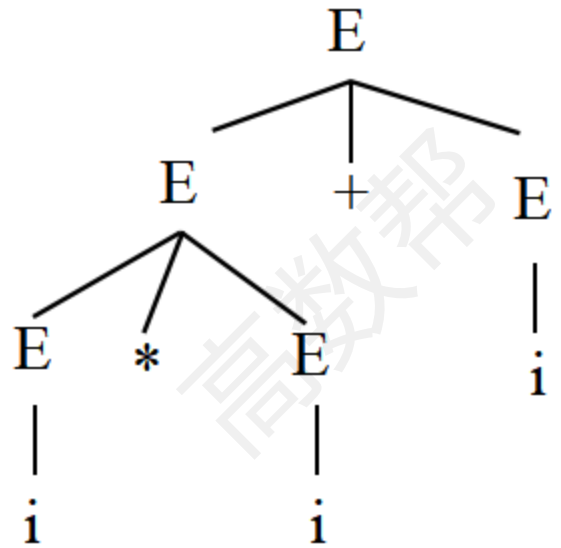
E*i+i

E*E+i

E+i

E+E

E



二、语法分析的方法

2.自上而下分析法

基本思想：它从文法的开始符号出发，反复使用各种产生式，寻找"匹配"的**推导**。

递归下降分析法：对每一语法变量(非终结符)构造一个相应的子程序，每个子程序识别一定的语法单位，通过子程序间的信息反馈和联合作用实现对输入串的识别。

预测分析程序

优点：直观、简单和易于手工实现。

三、自上而下分析面临的问题

1.自上而下就是从文法的开始符号出发，向下推导，推出句子。

1)带“回溯”的

2)不带回溯的递归子程序(递归下降)分析方法。

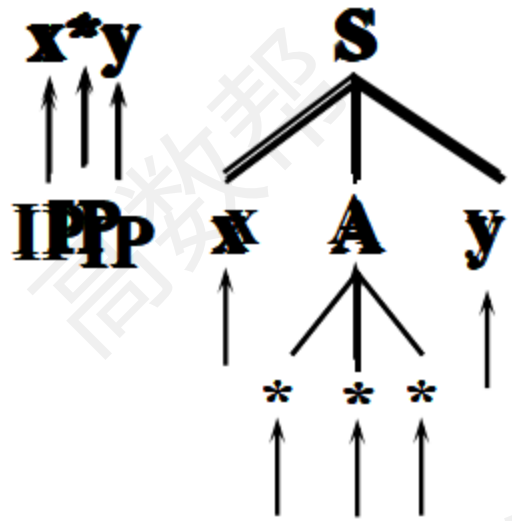
2.自上而下分析的主旨：对任何输入串，试图用一切可能的办法，从文法开始符号(根结点)出发，自上而下地为输入串建立一棵语法树。或者说，为输入串寻找一个最左推导。

【题2】假定有文法G(S):

(1) S→xAy

(2) A→**|*

分析输入串x*y(记为α)。



当某个非终结符有多个产生式候选时，可能带来如下问题:

1. 分析过程中，当一个非终结符用某一个候选匹配成功时，这种匹配可能是暂时的。出错时，不得不“回溯”。
2. 文法左递归问题。一个文法是含有左递归的，如果存在非终结符$P \rightarrow P\alpha \mid \beta$

含有左递归的文法将使自上而下的分析陷入无限循环。

4.2 LL(1)文法的判定

构造不带回溯的自上而下分析算法

1.要消除文法的左递归性

2.克服回溯

4.3 消除左递归

直接消除见诸于产生式中的左递归：假定关于非终结符P的规则为

$$P \rightarrow P\alpha \mid \beta$$

（其中β不以P开头）

我们可以把P的规则等价地改写为如下的非直接左递归形式：

$$P \rightarrow \beta P'$$

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

左递归变右递归

一般而言，假定关于P的全部产生式是

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

其中，每个α都不等于ε，每个β都不以P开头

那么，消除P的直接左递归性就是把这些规则改写成：

$$P \rightarrow \beta_1 P' \mid \beta_2 P' \mid \ldots \mid \beta_n P'$$

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

【题3】文法G(E):

$E \rightarrow E + T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid i$

经消去直接左递归后变成：

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

公式：

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

$P \rightarrow \beta_1 P' \mid \beta_2 P' \mid \ldots \mid \beta_n P'$

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

【题4】文法G(S):

$S \rightarrow Qc|c$

$Q \rightarrow Rb|b$

$R \rightarrow Sa|a$

虽没有直接左递归，但S、Q、R都是左递归的

$$S \Rightarrow Qc \Rightarrow Rbc \Rightarrow Sabc$$

一个文法消除左递归的条件：

➡ 不含以ε为右部的产生式

➡ 不含回路

一、消除左递归的算法

1. 把文法G的所有非终结符按任一种顺序排列成P_1，P_2，…，P_n；按此顺序执行；

2. FOR i:=1 TO n DO

   BEGIN

   FOR j:=1 TO i-1 DO

   把形如$P_i \rightarrow P_j\gamma$的规则改写成

   $P_i \rightarrow \delta_1\gamma|\delta_2\gamma|\ldots|\delta_k\gamma$;

   (其中$P_j \rightarrow \delta_1|\delta_2|\ldots|\delta_k$是关于$P_j$的所有规则)

   消除关于P_i规则的直接左递归性

   END

3. 化简由2所得的文法。去除那些从开始符号出发永远无法到达的非终结符的产生规则。

二、消除左递归算法的注意事项

1. 此算法适用于，消除了P→P 产生式和不以ε为右部的产生式的文法。

2. 该算法的第二步所做的工作可以理解为：

   1）若产生式出现直接左递归，则用消除直接左递归的方法消除；

   2）若产生式右部的最左符号是非终结符且序号大于左部的非终结符，则不作替换处理；

   3）若序号小于左部的非终结符， 则将序号小的非终结符（已处理过）用其右部串替换，然后消除新的直接左递归；

   4）因此每次替换的非终结符均为前面已经处理过的非终结符。

【题5】考虑文法G(S)

S→Qc|c

Q→Rb|b

R→Sa|a

令它的非终结符的排序为R、Q、S。

对于R，不存在直接左递归。 R→Sa|a产生式中S的序号大于R，不作替换处理

对于Q， Q→Rb|b中R的序号小于Q，把R代入到Q的有关候选后，把Q的规则变为

$$Q \rightarrow Sab \mid ab \mid b$$

不存在直接左递归，并且产生式中S的序号大于Q，不再做替换处理。

【题6】考虑文法G(S)

S→Qc|c

Q→Rb|b

R→Sa|a

令它的非终结符的排序为R、Q、S。

Q的规则变为

$$Q \rightarrow Sab \mid ab \mid b$$

对于S，S→Qc|c产生式中Q序号小于S，把Q的产生式代入到S的有关候选后，S变成

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

【题7】考虑文法G(S)

$S \rightarrow Qc|c$

$Q \rightarrow Rb|b$

$R \rightarrow Sa|a$

S变成

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

出现左递归，消除S的直接左递归后：

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

$S' \rightarrow abcS' \mid \varepsilon$

$Q \rightarrow Sab \mid ab \mid b$

$R \rightarrow Sa|a$

【题8】考虑文法G(S)

$S \rightarrow Qc|c$

$Q \rightarrow Rb|b$

$R \rightarrow Sa|a$

消除S的直接左递归后：

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

$S' \rightarrow abcS' \mid \varepsilon$

$Q \rightarrow Sab \mid ab \mid b$

$R \rightarrow Sa|a$

关于Q和R的规则已是多余的，化简为：

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

$S' \rightarrow abcS' \mid \varepsilon$

注意，由于对非终结符排序的不同，最后所得的文法在形式上可能不一样。但不难证明，它们都是等价的。

比如：若对文法(4.3)的非终结符排序选为S、Q、R，那么，最后所得的无左递归文法是：

$$S \rightarrow Qc \mid c$$

$$Q \rightarrow Rb \mid b$$

$$R \rightarrow bcaR' \mid caR' \mid a\,R'$$

$$R' \rightarrow bca\,R' \mid \varepsilon$$

4.4 FIRST & FOLLOW集的计算

一、消除回溯、提左因子

产生回溯的原因：推导时，若产生式存在多个候选式，选择哪个进行推导存在不确定性。

为了消除回溯就必须保证：对文法的任何非终结符，当要它去匹配输入串时，能够根据它所面临的输入符号准确地指派它的一个候选去执行任务，并且此候选的工作结果应是确信无疑的。

令G是一个不含左递归的文法，对G的所有非终结符的每个候选α定义它的终结首符集FIRST(α)为：

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

特别是，若$a \overset{*}{\Rightarrow} \varepsilon$，则规定 **ε∈FIRST(α)**。

如果非终结符A的所有候选首符集两两不相交，即A的任何两个不同候选α_i和α_j

$$\mathbf{FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi}$$

当要求A匹配输入串时，A就能根据它所面临的第一个输入符号a，准确地指派某一个候选前去执行任务。这个候选就是那个终结首符集含a的α。

提取公共左因子:

假定关于A的规则是

$$A \rightarrow \delta\beta_1 | \delta\beta_2 | \ldots | \delta\beta_n | \gamma_1 | \gamma_2 | \cdots | \gamma_m$$

(其中，每个γ不以δ开头)

那么，可以把这些规则改写成

$$A \rightarrow \delta A' | \gamma_1 | \gamma_2 | \cdots | \gamma_m$$

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

经过反复提取左因子，就能够把每个非终结符(包括新引进者)的所有候选首符集变成为两两不相交。

4.5 LL(1)分析条件

假定S是文法G的开始符号，对于G的任何非终结符A，我们定义

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

特别是，若 S $\overset{*}{\Rightarrow}$ ...A ， 则规定

 # ∈FOLLOW(A)

一、构造不带回溯的自上而下分析的文法条件

1. 文法不含左递归；

2. 对于文法中每一个非终结符A的各个产生式的候选首符集两两不相交。即：

   若$A\rightarrow\alpha_1|\alpha_2|\ldots|\alpha_n$则 $FIRST(\alpha_i)\cap FIRST(\alpha_j)=\phi \quad (i\neq j)$

3. 对文法中的每个非终结符A，若它存在某个候选首符集包含ε，则

   $FIRST(\alpha_i)\cap FOLLOW(A)=\phi \quad (i=1,2,...,n)$

如果一个文法G满足以上条件，则称该文法G为LL(1)文法。

对于一个满足上述条件的文法，可以对其输入串进行有效的无回溯的自上而下分析。假设要用非终结符A进行匹配，面临的输入符号为a，A的所有产生式为

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

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

2. 若a不属于任何一个候选首符集，则：

   (1) 若ε属于某个$FIRST(\alpha_i)$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。

二、构造FIRST(α)

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

对每一文法符号$X \in V_T \cup V_N$构造FIRST(X)

连续使用下面的规则，直至每个集合FIRST不再增大为止：

1. 若$X \in V_T$，则FIRST(X)＝{X}。

2. 若$X \in V_N$，且有产生式$X \to a\ldots$，则把a加入到FIRST(X)中；若$X \to \varepsilon$也是一条产生式，则把ε也加到FIRST(X)中。

3.若$X \to Y\ldots$是一个产生式且$Y \in V_N$，则把FIRST(Y)中的所有非ε-元素都加到FIRST(X)中；

若$X \to Y_1 Y_2 \ldots Y_k$是一个产生式，Y_1，…，Y_{i-1}都是非终结符，而且，对于任何j，$1 \le j \le i-1$，$FIRST(Y_j)$都含有ε(即$Y_1 \ldots Y_{i-1} \varepsilon$)，则把$FIRST(Y_i)$中的所有非ε-元素都加到FIRST(X)中；特别是，若所有的$FIRST(Y_j)$均含有ε，j＝1，2，…，k，则把ε加到FIRST(X)中。

对文法G的任何符号串$\alpha=X_1X_2\ldots X_n$构造集合FIRST(α)。

1. 置FIRST(α)＝FIRST(X_1)\{ε}；

2. 若对任何$1\le j\le i-1$，$\varepsilon\in$FIRST(X_j)，则把FIRST(X_i)\{ε}加至FIRST(α)中；特别是，若所有的FIRST(X_j)均含有ε，$1\le j\le n$，则把ε也加至FIRST(α)中。

显然，若$\alpha=\varepsilon$则FIRST(α)＝{ε}。

视频讲解更清晰

三、构造FOLLOW(A)

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

对于文法G的每个非终结符A构造FOLLOW(A)的办法是，连续使用下面的规则，直至每个FOLLOW不再增大为止：

1. 对于文法的开始符号S，置＃于FOLLOW(S)中；

2. 若$A\rightarrow\alpha B\beta$是一个产生式，则把$FIRST(\beta)\backslash\{\varepsilon\}$加至FOLLOW(B)中；

3. 若$A\rightarrow\alpha B$是一个产生式，或$A\rightarrow\alpha B\beta$是一个产生式而$\beta \Longrightarrow \varepsilon$

(即$\varepsilon\in FIRST(\beta)$)，则把FOLLOW(A)加至FOLLOW(B)中。

总结：FIRST看左，FOLLOW看右

【题9】对于文法G(E)

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

构造每个非终结符的FIRST和FOLLOW集合：

FIRST(E) ={(,i}　　FOLLOW(E) ={),#}

FIRST(E′)={+, ε}　　FOLLOW(E′)={),#}

FIRST(T) ={(,i}　　FOLLOW(T) ={+,),#}

FIRST(T′)={*, ε}　　FOLLOW(T′)={+,),#}

FIRST(F) ={(,i}　　FOLLOW(F) ={*,+,),#}

4.7 递归下降分析程序构造

1.构造不带回溯的自上而下分析程序：1）要消除文法的左递归性　2）克服回溯

2.构造不带回溯的自上而下分析器

3.分析程序由一组递归过程组成，文法中每个非终结符对应一个过程；所以这样的分析程序称为递归下降分析器。(因为文法的定义通常是递归的)

4.几个全局过程和变量：

- ADVANCE，把输入串指示器IP指向下一个输入符号，即读入一个单字符号
- SYM，IP当前所指的输入符号
- ERROR，出错处理子程序

【题10】文法G(E):

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

视频讲解更清晰

每个非终结符有对应的子程序的定义，首先在分析过程中，当需要从某个非终结符出发进行展开(推导)时，就调用这个非终结符对应的子程序。

【题11】文法G(E):

E→TE′

E′→+TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | i

对应的递归下降子程序为:

```
PROCEDURE  E;
BEGIN
                T;  E′
END;
```

```
PROCEDURE  E′;
          IF  SYM='+'
THEN
          BEGIN
          ADVANCE;
                    T;  E′
          END
```

【题12】文法G(E):

E→TE′

E′→+TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | i

对应的递归下降子程序为:

```
PROCEDURE  T;
BEGIN
          F;  T′
END
```

```
PROCEDURE  T′;
IF SYM='*' THEN
BEGIN
   ADVANCE;
   F;  T′
END;
```

【题13】文法G(E):

E→TE′

E′→+TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | i

对应的递归下降子程序为:

```
PROCEDURE  F;
IF SYM='i' THEN ADVANCE
ELSE
  IF SYM='(' THEN
  BEGIN
       ADVANCE;
    E;
    IF SYM=')'  THEN  ADVANCE
         ELSE ERROR
  END
  ELSE ERROR;
```

4.4 预测分析程序

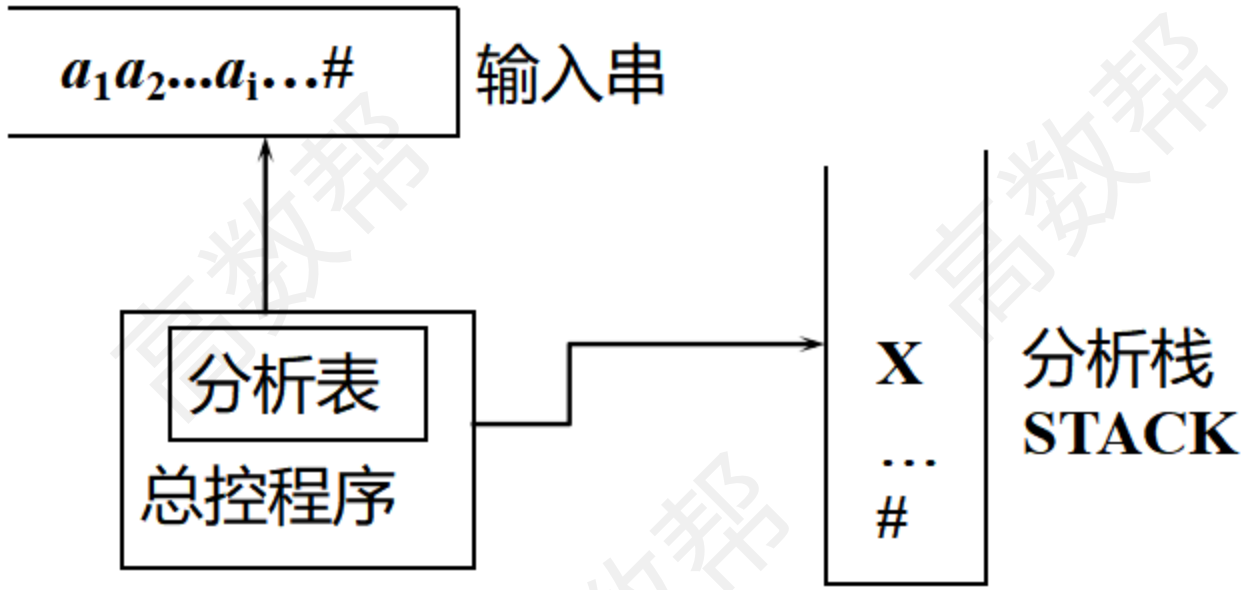
一、预测分析程序工作原理

1.预测分析程序或LL(1)分析法：

1） 总控程序

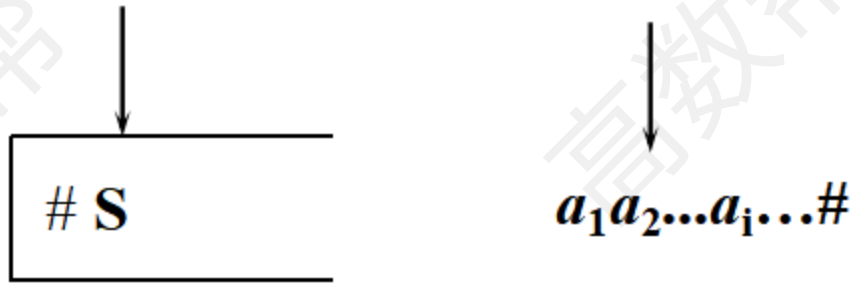
2）分析表 M[A，a]矩阵，$A \in V_N$，$a \in V_T$是终结符或‘#’

3）分析栈 STACK 用于存放文法符号



预测分析程序的工作图

分析开始时：



总控程序根据现行栈顶符号X和当前输入符号a，执行下列三种动作之一:

1. 若X＝a＝‘#’，则宣布分析成功，停止分析。

2. 若X＝a≠‘#’，则把X从STACK栈顶逐出，让a指向下一个输入符号。

3. 若X是一个非终结符，则查看分析表M。

1）若M[X，a]中存放着关于X的一个产生式，把X逐出STACK栈顶，把产生式的右部符号串按反序一一推进STACK栈(若右部符号为ε，则意味不推什么东西进栈)。在把产生式的右部符号推进栈的同时应做这个产生式相应的语义动作。

2）若M[X，a]中存放着“出错标志”，则调用出错诊察程序ERROR。

预测分析程序的总控程序：

```
BEGIN
    首先把‘#’然后把文法开始符号推进STACK栈;
    把第一个输入符号读进a;
    FLAG:=TRUE;
    WHILE  FLAG  DO
  BEGIN
          把STACK栈顶符号上托出去并放在X中;
          IF X∈V_T THEN
              IF X= a  THEN 把下一输入符号读进a
                              ELSE   ERROR
```

匹配成功

分析成功

ELSE IF X=‘#’ THEN

IF X=a THEN FLAG:=FALSE

ELSE ERROR

ELSE IF M[X,a]={X→$X_1X_2…X_k$}THEN

把$X_k,X_{k-1},…,X_1$一一推进STACK栈

/* 若$X_1X_2…X_k$=ε，不推什么进栈 */

ELSE ERROR

END OF WHILE;

STOP /*分析成功，过程完毕*/

END

推导

【题14】对于文法G(E)

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

输入串为$i_1*i_2+i_3$，利用分析表进行预测分析：

步骤	符号栈	输入串	所用产生式
0	#E	$i_1*i_2+i_3$#	
1	#E′T	$i_1*i_2+i_3$#	E→TE′
2	#E′T′F	$i_1*i_2+i_3$#	T→FT′
3	#E′T′i	$i_1*i_2+i_3$#	F→i

视频讲解更清晰

步骤	符号栈	输入串	所用产生式
3	#E′T′i	$i_1*i_2+i_3$#	F→i
4	#E′T′	$*i_2+i_3$#	
5	#E′T′F*	$*i_2+i_3$#	T′→*FT′
6	#E′T′F	i_2+i_3#	
7	#E′T′i	i_2+i_3#	F→i

步骤	符号栈	输入串	所用产生式
7	#E′T′i	i_2+i_3#	F→i
8	#E′T′	$+i_3$#	
9	#E′	$+i_3$#	T′→ε
10	#E′T+	$+i_3$#	E′→+TE′
11	#E′T	i_3#	

步骤	符号栈	输入串	所用产生式
11	#E′T	i_3#	
12	#E′T′F	i_3#	T→FT′
13	#E′T′i	i_3#	F→i
14	#E′T′	#	
15	#E′	#	T′→ε
16	#	#	E′→ε

2.分析表M[A，a]的构造

在对文法G的每个非终结符A及其任意候选α都构造出FIRST(α)和FOLLOW(A)之后，现在可以用它们来构造G的分析表M[A，*a*]。

1. 对文法G的每个产生式A→α执行第2步和第3步；
2. 对每个终结符*a* ∈FIRST(α)，把A→α加至M[A，*a*]中；
3. 若ε∈FIRST(α)，则对任何b∈FOLLOW(A)把A→α加至M[A，b]中。
4. 把所有无定义的M[A，*a*]标上“出错标志”。

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

FIRST(E) ={(,i}

FIRST(E′)={+, ε}

FIRST(T) ={(,i}

FIRST(T′)={*, ε}

FIRST(F) ={(,i}

FOLLOW(E) ={),#}

FOLLOW(E′)={),#}

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

FOLLOW(T′)={+,),#}

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

文法G[E]的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)		

如果G是左递归或二义的，那么，M至少含有一个多重定义入口。因此，消除左递归和提取左因子将有助于获得无多重定义的分析表M。

可以证明，一个文法G的预测分析表M不含多重定义入口，当且仅当该文法为LL(1)的。

LL(1) 中的第一个“ L” 意味着自左而右地扫描输入 ,第二个“ L”意味着生成一个最左推导,并且“1”意味着为做出分析动作的决定，在每一步利用一个向前看符号。