编译技术

计算机学院计算机系

刘佰龙

liubailong@cumt.edu.cn

04 语法分析

4 语法分析

- 4.1 语法分析器的作用
- 4.2 书写标准文法
- 4.3 自顶向下语法分析
- 4.4 自底向上语法分析
- 4.5 LALR(1)分析程序的生成器YACC

语法分析方法

1 自顶向下分析

A）递归下降分析法:

包含回溯的递归下降分析法

不含回溯的递归下降分析法(递归预测分析法)

B）非递归的预测分析法:LL分析法

2 自底向上分析

A）算符优先分析法

B）LR分析法

递归下降分析法

（1）包含回溯的递归下降分析法

（2）不含回溯的递归下降分析法(递归预测分析法)

包含回溯的递归下降分析法

思想：从语法的开始符号出发，试探使用不同产生式，寻找匹配于输入符号串的推导。或者说，从对应文法开始符号的根结点出发，自顶向下为输入符号串建立一棵分析树。

这种分析方法实质上是一种试探过程，是因为文法中，对于某个非终结符号的规则其右部有多个选择，并根据所面临的输入符号不能准确的确定所要选择时，就可能出现回溯。

例子

S →cAd

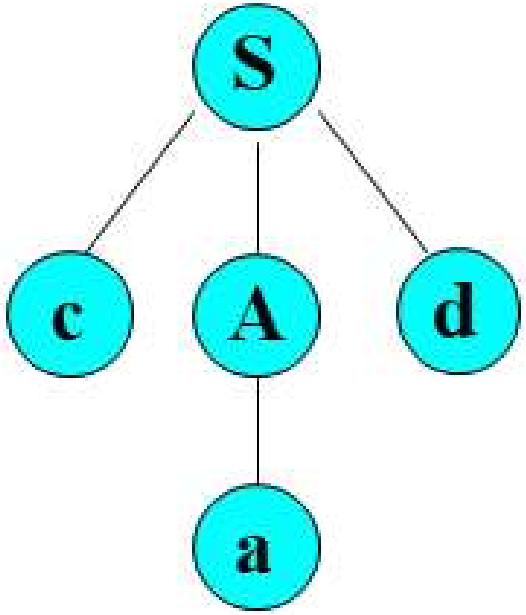
A →ab | a

输入符号串cad是否为文法的句子?

c	a	d

S → cAd

A → a



输入符号串cad合法

缺点

上述分析称为带回溯的自顶向下分析。对于w，从文法的开始符号出发，反复使用不同的产生式谋求匹配输入串。当用某个非终结符号的候选式进行匹配失败时，删除这个失败分析建立的分析树分支并回头查输入符号，以便与其它候选式匹配。

这种带回溯的自顶向下分析技术，效率很低，代价极高，因此，它只有理论上的意义。

不含回溯的递归下降分析法

在不含左递归和每个非终结符的所有候选式的终结首符集都两两不相交条件下，构造一个不带回溯的自上而下分析程序，该分析程序由一组递归过程组成，每个过程对应文法的一个非终结符。

这样的一个分析程序称为递归预测分析器。

在推导过程中，根据当前的向前看符号，决定选择哪个产生式往下推导，因此，分析过程是完全确定的。这种分析称为自顶向下的预测分析。

终结首符集辅助确定候选式

若当前符号 = a，对 A 展开，而 $A \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_m$，

那么，要知道哪一个α_i是获得以a开头的串的唯一替换式。

即：选择哪一个α_i，亦即通过查看第一个（当前）符号来发现合适的替换式α_i。

怎样选择α_i?

①以a为头的α_i

②如有多个α_i以a为头的，这是文法的问题

终结首符集辅助确定候选式

定义FIRST(α)为α的终结首符集：

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

若$\alpha \overset{*}{\Rightarrow} \varepsilon$，规定$\varepsilon \in FIRST(\alpha)$

若有$A \rightarrow \alpha$，且$\alpha \overset{+}{\Rightarrow} a\ldots$，当遇到输入字符为a时，选择此候选式替换栈中的A

例子

已知：$P \rightarrow \alpha$；$F \rightarrow a...|b...|\varepsilon$；$Q \rightarrow c...|\varepsilon$, 其中$\alpha$=FQ。

则First(α)= First(FQ)

（1）First(F) ⊆ First(FQ)

（2）当$F \rightarrow \varepsilon$时，则 α = FQ → Q，因此First(Q) ⊆ First(FQ)

故，

$$First(FQ) = First(F) \cup First(Q)$$
$$= \{a,b,c,\varepsilon\}$$

定理

若一个 $A \in V_N$ 有许多 $FIRST(\alpha_i)$，如果A的任何两个候选式 α_i 和 α_j，均有

$$FIRST(\alpha_i) \cap FIRST(\alpha_j) = \varphi$$

意味着，A的每一候选式α推导后所得的字符串第一个 V_T 均不同。

于是，对输入符号a，如果 $a \in FIRST(\alpha_i)$，则 $a \text{ not} \in FIRST(\alpha_j)$，$(j \neq i)$，因此，对A的展开无疑应选候选式 α_i，才有可能命中。

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**就提示我们哪一个替换式是仅有可能成功的替换式。

- 抽象地看问题：

若要求不得回溯，文法G（当然G不含左递归）的必要条件是什么，也即对文法有什么要求？

FIRST(α_i) ∩ FIRST(α_j) ≠ φ? ?

提取公共左因子

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

(其中,每个γ不以δ开头)

那么,可以把这些规则改写成

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

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

例

S →cAd

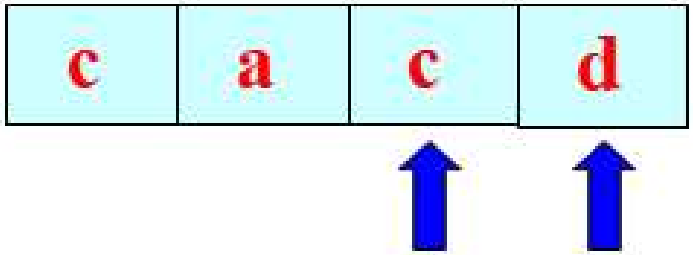
A →ab | ac

输入符号串cacd是否为文法的句子？

S →cAd

A →aB

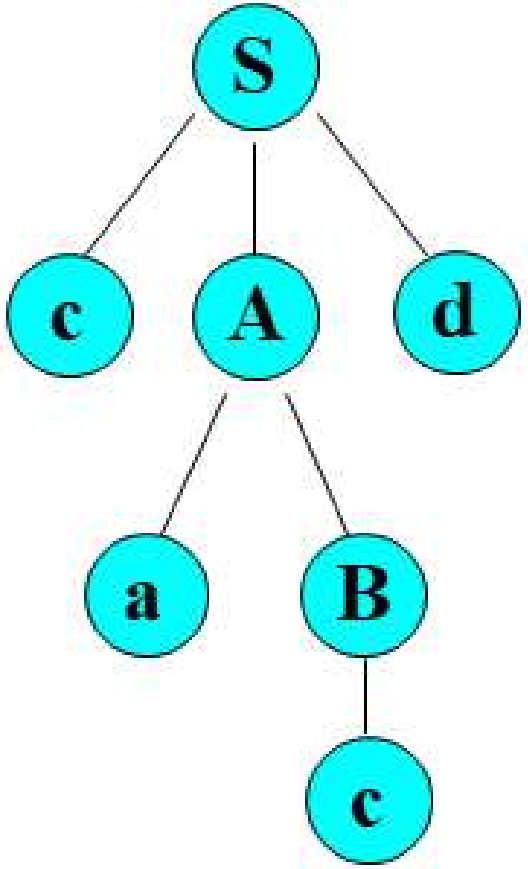
B →b | c



S → cAd

A → aB

B → c



输入符号串cacd合法

例

文法：*type* →*simple*

| array [*simple*] of *type*

simple →integer

| char

| num dotdot num

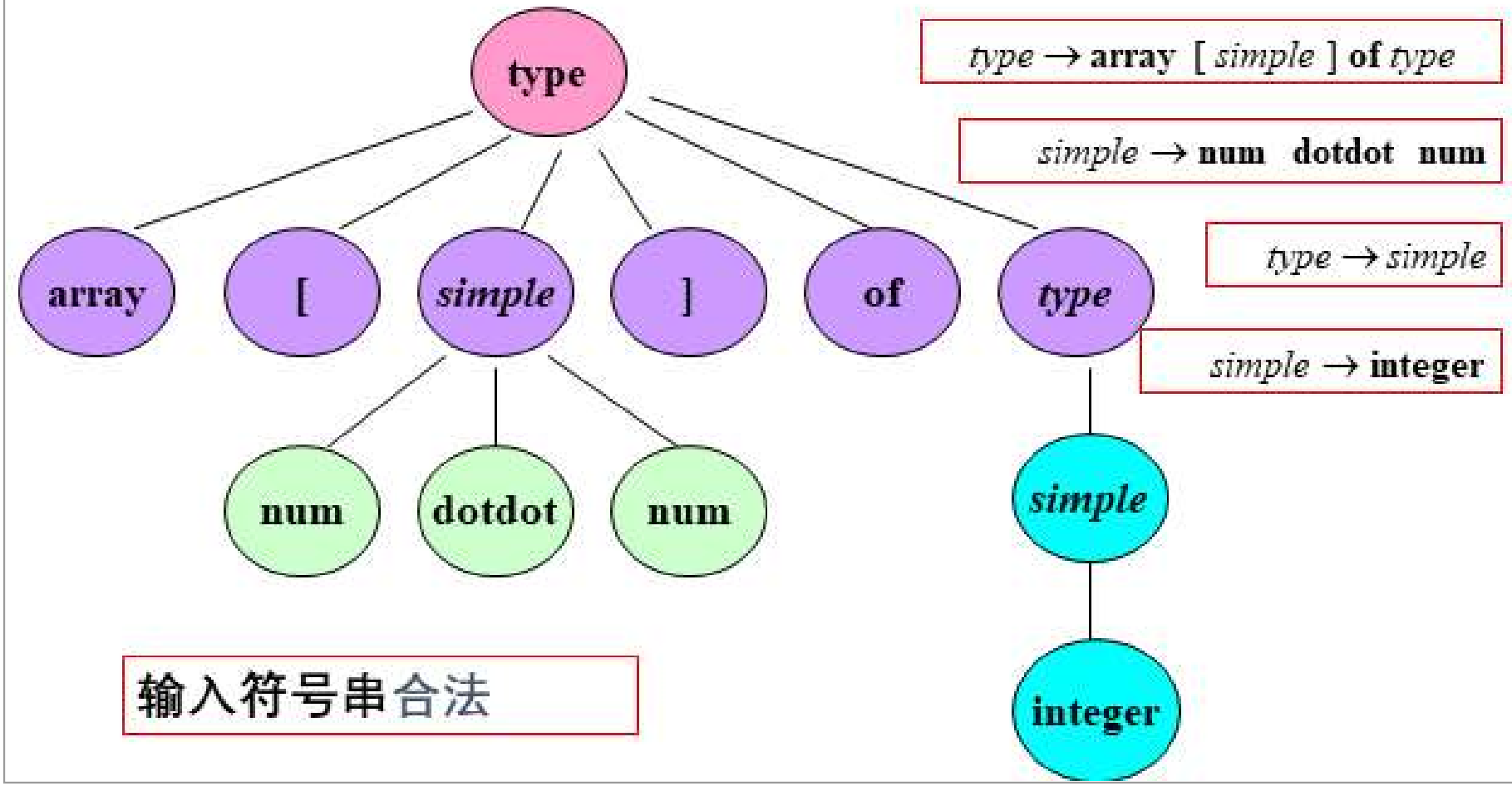
输入符号串：

array [num dotdot num] of integer是否为该文法的句子？

对文法先进行消除左递归并提取公共左因子

例

array	[	num	dotdot	num	]	of	integer



递归的预测分析法

- 递归的预测分析法是指：在递归下降分析中，对文法中的每个非终结符编写一个函数或子程序，其功能是识别由该终结符所表示的语法成分，由于语言的文法常常是递归定义的，因此这组函数或子程序必须以递归方式调用。

```
void A( ) {
1)   选择一个A产生式，A→X1X2 …Xk ；
2)   for (i = 1 to k) {
3)       if (Xi是一个非终结符号)
4)           调用过程Xi( )；
5)       else if(Xi 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 */;
     }
  }
```

递归下降分析器

- 定义全局过程和变量
  - ADVANCE，把输入串指示器IP指向下一个输入符号，即读入一个单词符号
  - SYM，IP当前所指的输入符号
  - ERROR，出错处理子程序

例

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

每个非终结符所对应的递归子程序如下所示：

例

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

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

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

$E \rightarrow TE'$　　$T \rightarrow FT'$　　$E' \rightarrow +TE' | \varepsilon$

ADVANCE：读入下一个单词

例

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

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

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

$F \rightarrow (E)|\ i$

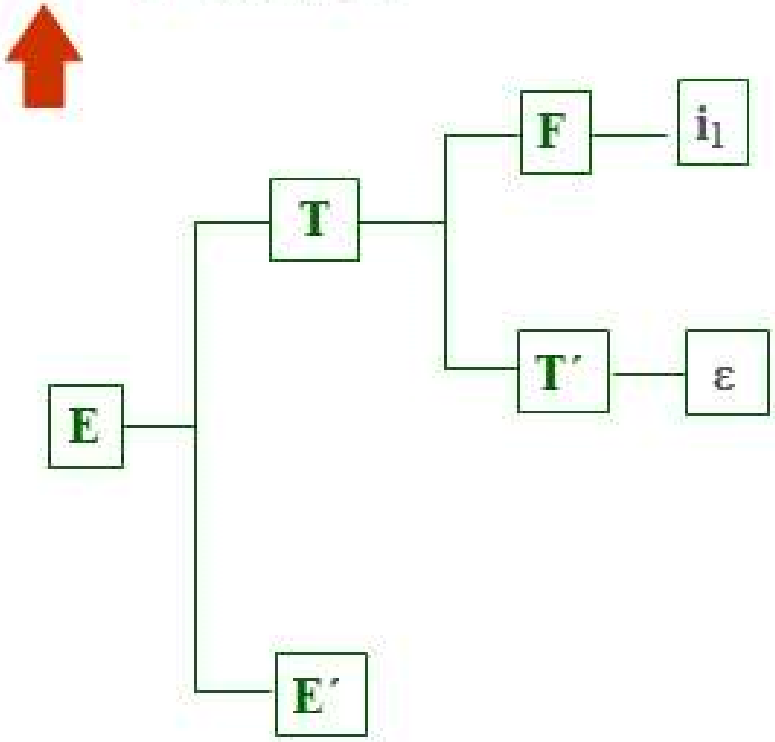
面临输入：$i_1+i_2*i_3$时的分析步骤如下：

递归下降子程序设计

- 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

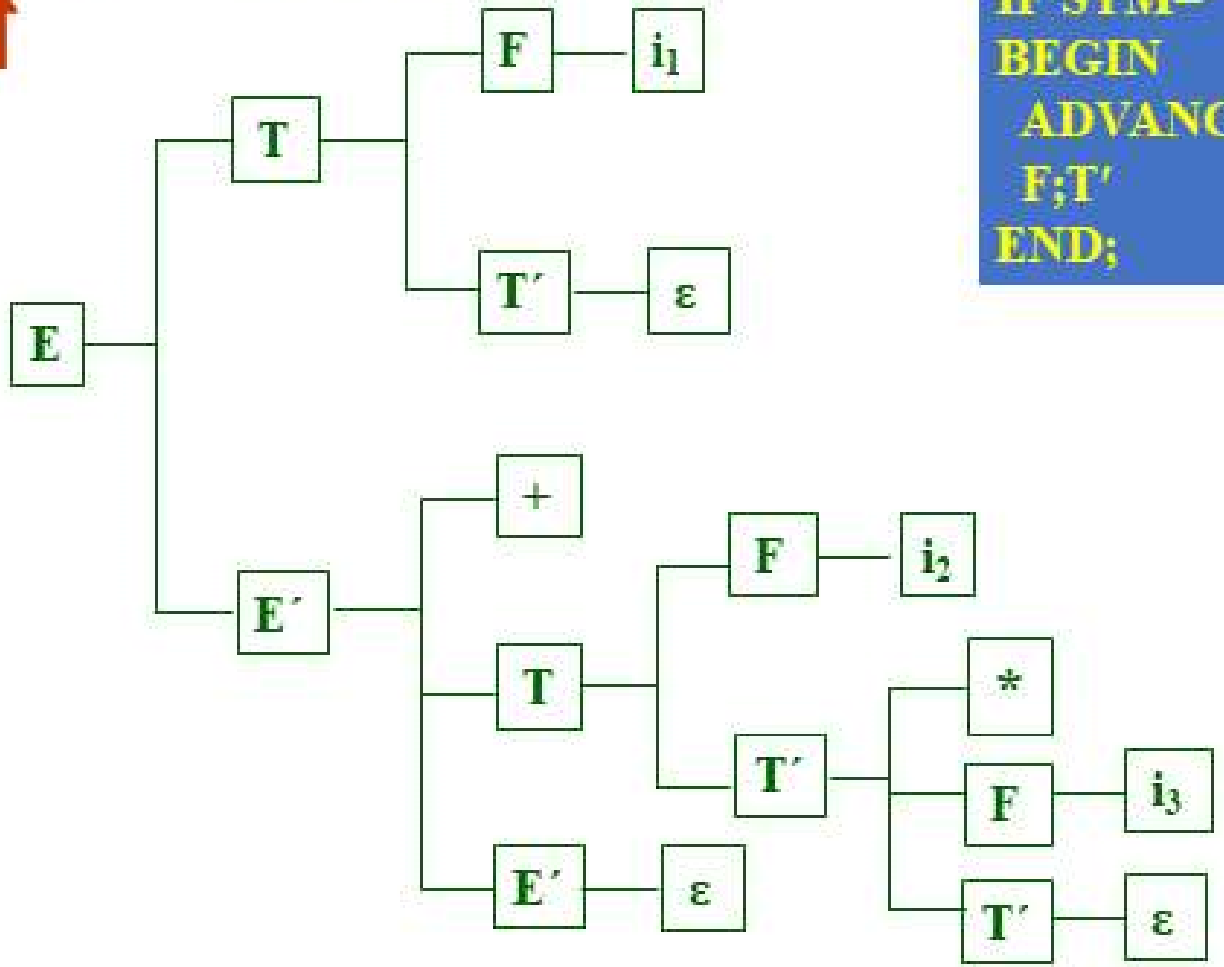
```
主程序:
PROGRAM PARSER;
BEGIN
    ADVANCE;
    E;
END;
```

$i_1 + i_2 * i_3$ 分析过程：




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

i_1 + i_2 * i_3 分析过程：




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

➢+id+id什么时候会报错呢？

注意

- 有ε，自动匹配，不会失败
- 无成功/失败消息返回
- 出错位置不确切

递归预测分析器的构造

$S \rightarrow ABC$　　$A \rightarrow a$　　$B \rightarrow b$　　$C \rightarrow c$

```
Void S(){
A();
B();
C();
}
```

```
Void A()
{
   if(ch=='a') n++;
   else{error;}
}
```

```
Void B()
{
  if(ch=='b')    n++;
  else{error;}
}
```

```
Void C()
{
   if(ch=='c') n++;
   else{error;}
}
```

递归预测分析器的构造

S → ABC　　A → a | xB　　B → b　　C → c

```
Void S(){
A();
B();
C();
}
```

```
Void A(){
if(ch=='a') n++;
else{
 if(ch=='x')
      {n++;B();}
 else
      {error;}
  }
}
```

```
Void B(){
   if(ch=='b'){n++;}
   else{error;}
}
```

```
Void C(){
     if(ch=='c')
       {n++;}
     else{error;}
}
```

例

$S \rightarrow ABC|BS \quad A \rightarrow a|x \quad B \rightarrow b \quad C \rightarrow c$

S有两个候选产生式，它的代码如何写?

根据向前看符号来确定，即first(ABC)和first(BS)确定

递归预测分析器的构造

（1）向前看符号决定了选择哪个产生式右部的文法符号串与输入符号串匹配，在程序中由向前看符号可以确定一段程序完成与某一产生式右部的匹配。

（2）每一个非终结符号对应一个函数，函数的功能是根据向前看符号扩展分析树，同时用该非终结符号的儿子结点匹配输入符号串。

非递归预测分析方法的引入

- **问题：** 用递归子程序描写递归下降分析器，要求实现该编译系统的语言允许递归。

- **改进：** 使用一张分析表和一个栈同样可实现递归下降分析，用这种方法实现的语法分析程序叫**预测分析程序**。

预测分析表

预测分析表是矩阵M[A,a]，其中行标A是非终结符V_n，列标a是终结符V_t或串终结符（#），矩阵元素M[A,a]是存放A的一个候选式，指出当前栈顶符号为A且读入符号为a时应选的候选式，或者存放“错误标志”，指出A不该面临输入符号。

预测分析表：

$$M[A,a]=\begin{cases} A \rightarrow \alpha_i \\ \text{或} \\ \text{error} \end{cases}$$

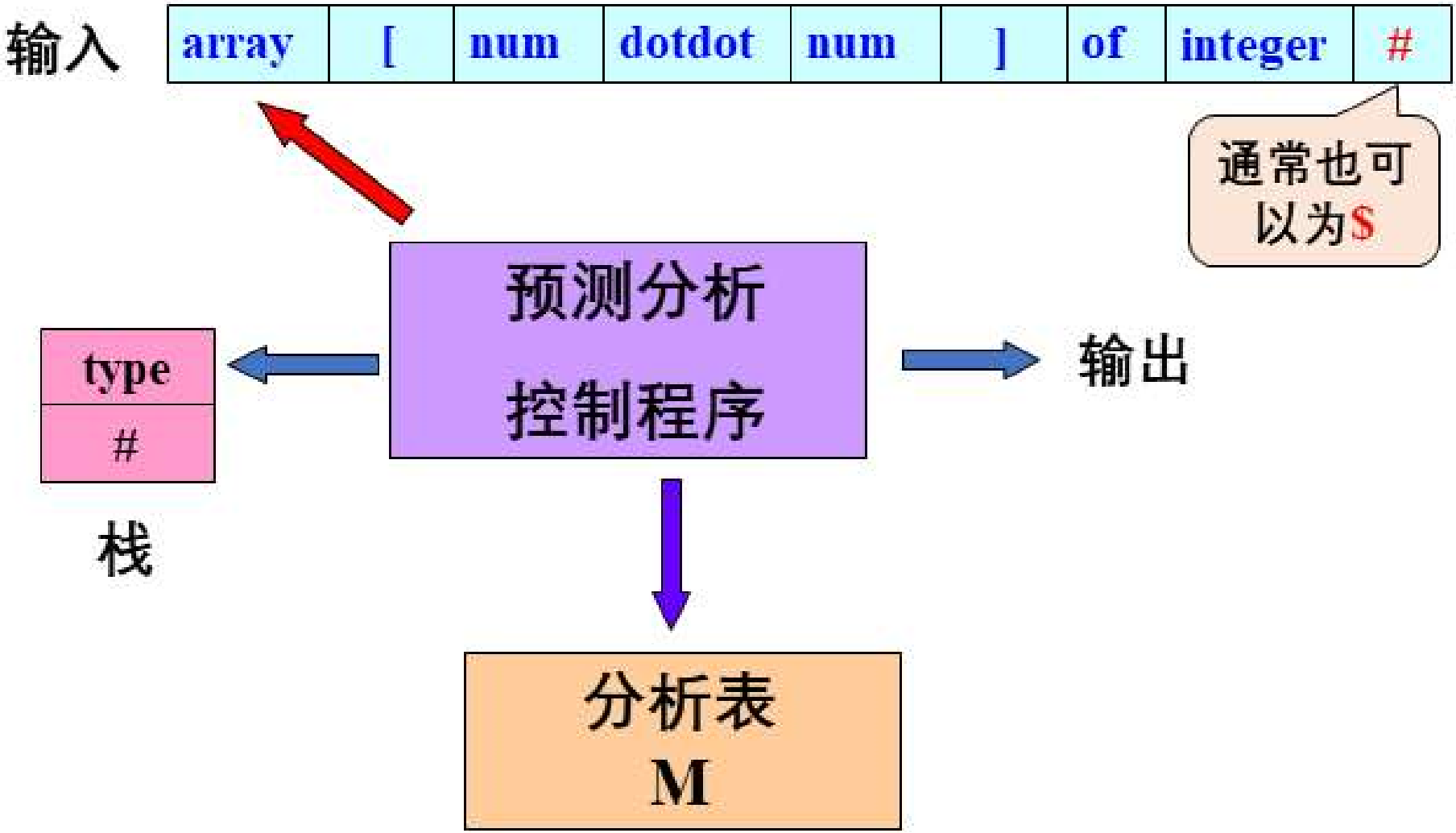
$A \in V_n$　　$\alpha_i \in V^*$　　$a \in V_t$ or #

如何构造，之后介绍

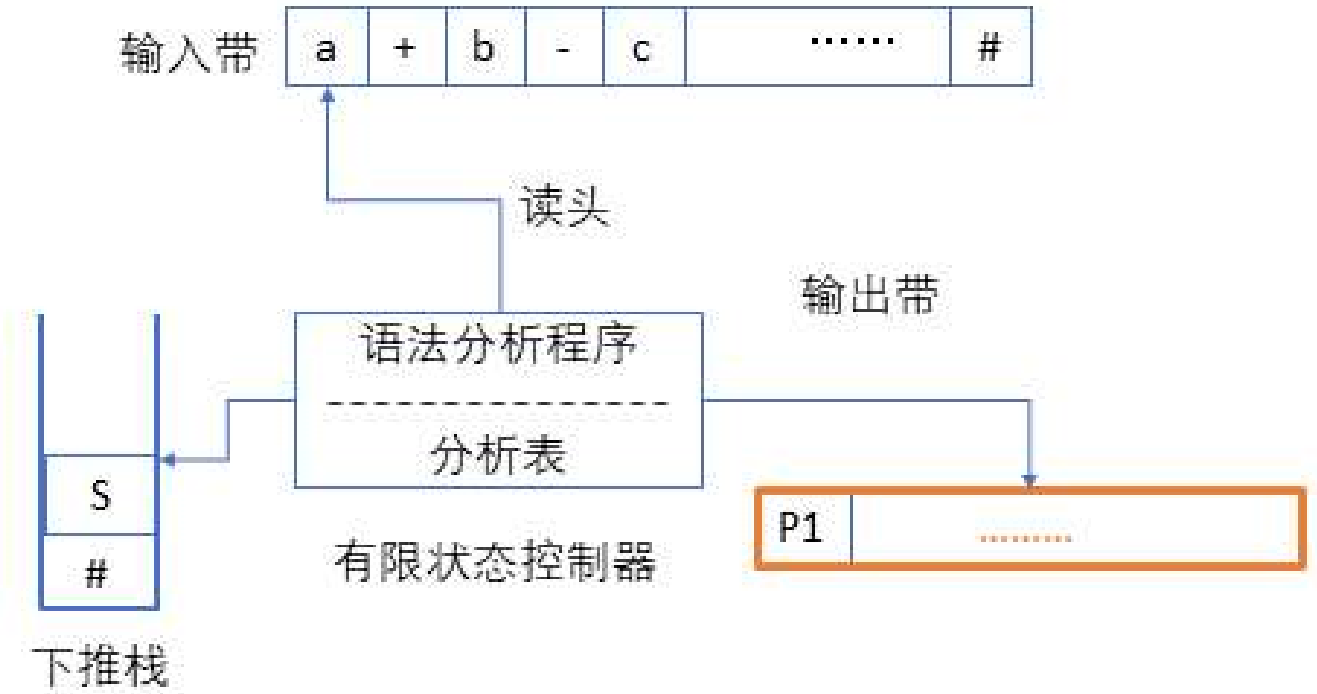
非递归的预测分析法

- 非递归预测分析器的模型
- 非递归预测分析器的分析演示
- 计算 FIRST和 FOLLOW集合
- 非递归预测分析表的构造
- LL(1)文法
- 在预测分析法中的错误处理

非递归预测分析器的模型



非递归预测分析器的模型



分析表

（1）分析表：二维矩阵M

$$M[A,a]=\begin{cases} A \rightarrow \alpha_i \\ 或 \\ error \end{cases}$$

$A \in V_n$　　$\alpha_i \in V^*$　　$a \in V_t$ or #

符号栈（下推栈）

开始状态

xxxxxx#

符号栈 #S

工作状态

a……#

符号栈 #…..X

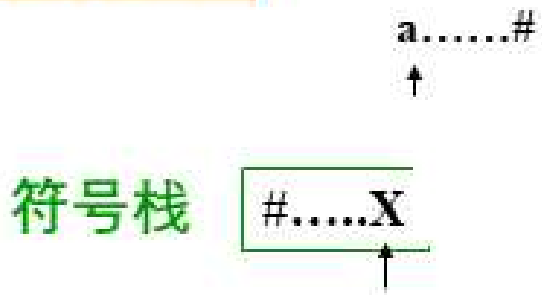
查分析表得：

$X \in V_n$, $M[X,a] = X \rightarrow \alpha_i$

$X \in V_t$, $X = a$

符号栈（下推栈）

出错状态

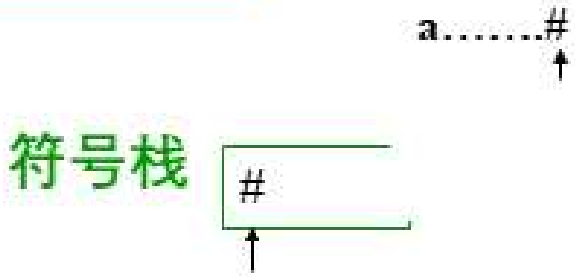


查分析表得：

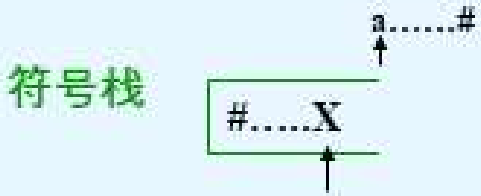
$X \in V_n$, $M[X,a]$= error

$X \in V_t$, $X \neq a$

结束状态



执行程序



主要实现如下操作:

- 把#和文法开始符号S推进栈,并读入输入串的第一个单词a,重复下述过程直到正常结束或出错.
- 根据栈顶文法符号X和当前单词符号a的关系进行不同的处理

(1)若X=a=#，分析成功，停止。匹配输入串成功.
(2)若X=a≠#，把X从栈顶弹出，再读入下一个符号.
(3)若$X \in V_n$，查分析表M。

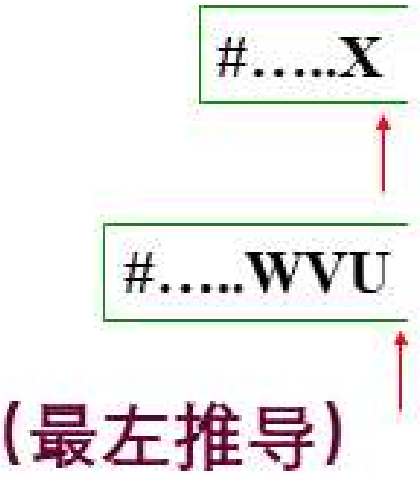
执行程序

若$X \in V_n$，查分析表M。

a) M[X,a]= X → UVW
则将X弹出栈，将UVW压入
注：U在栈顶

b) M[X, a] = error 转出错处理

c) M[X, a] = X → ε
则将X弹出栈,继续分析。



非递归预测分析器的分析演示

文法：*type* →*simple*

| ↑id

| array [*simple*] of *type*

simple →integer

| char

| num dotdot num

输入符号串：

array [num dotdot num] of integer是否为该文法的句子？

array	[	num	dotdot	num	]	of	integer	#

type → array [*simple*] of *type*

simple → num dotdot num

type → *simple*

simple → integer

句子array [num dotdot num] of integer 合法

栈	#	输入符号				
		↑	array	integer	char	num
非终结符	*type*	*type* → ↑id	*type* → array [*simple*] of *type*	*type* →*simple*	*type* →*simple*	*type* →*simple*
	simple	error	error	*simple* →integer	*simple* →char	*simple* →num dotdot num

预测分析控制程序

置 ip指向输入符号串w#的第一个符号；

REPEAT

令X为栈顶符号，a是 ip所指向的符号.

IF $(X \in V_T \cup \{\#\})$

IF (X= =a) { 弹出X; ip:=ip+1}

ELSE error( )

ELSE

IF $(M[x,a]=x \rightarrow y_1y_2\ldots y_k)$

{pop(st);

push(st,y_k);…push(st,y_1);

write($x \rightarrow y_1y_2\ldots y_k$) }

ELSE error( )

UNTIL (X= =#)

例子

文法：

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

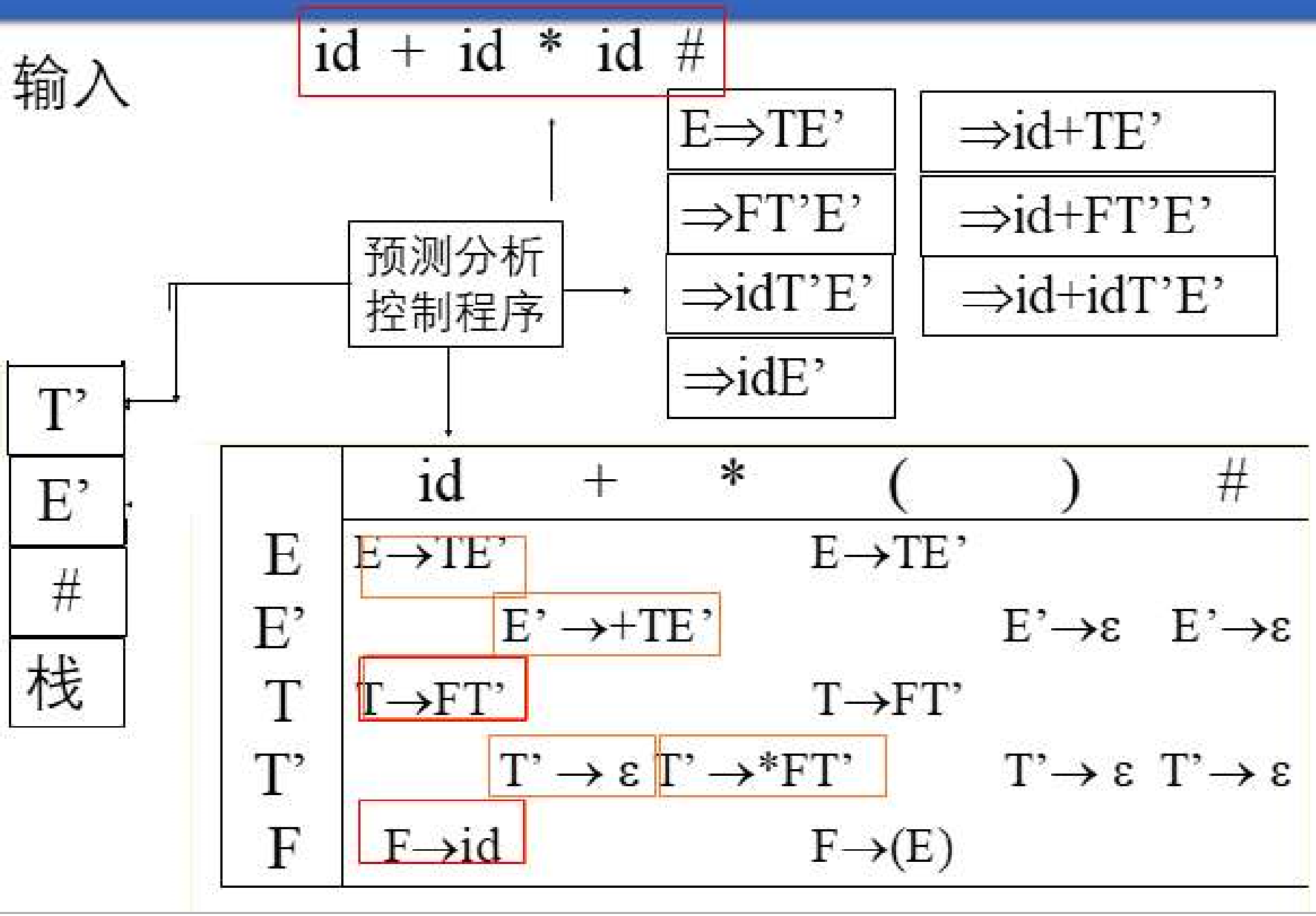
输入符号串 id+id*id 是否为该文法的句子？

例子

非终结符号	输入符号					
	id	+	*	(	)	#
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 →id			F →(E)		

文法分析表M

例子


输入
id + id * id #
预测分析
控制程序
E⇒TE’
⇒FT’E’
⇒idT’E’
⇒idE’
⇒id+TE’
⇒id+FT’E’
⇒id+idT’E’
T’
E’
#
栈
id + * ( ) #
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→id F→(E)

在输入id+id*id上预测分析器作出的移动

栈	输入	输出
#E	id+id*id #	
#E'T	id+id*id #	E →TE'
#E'T'F	id+id*id #	T →FT'
#E'T'id	id+id*id #	F →id
#E'T'	+id*id #	
#E'	+id*id #	T'→ε
#E'T+	+id*id #	E'→+TE'
#E'T	id*id #	
#E'T'F	id*id #	T →FT'
#E'T'id	id*id #	F→id
#E'T'	*id #	
#E'T'F*	*id #	T'→*FT'
#E'T'F	id #	
#E'T'id	id #	F→id
#E'T'	#	
#E'	#	T'→ε
#	#	E'→ ε

结论

- 输出的产生式对应了最左推导
- 栈中：残缺规范句型<未被匹配的句型>
- M[X,a]：指出V_N按哪一条产生式扩展，依赖于M表指出(栈顶，a)时，如何扩充语法树，出错了可以立即发现。

分析表的构造

◆分析表格式：

	id	+	*	(	)	#
E	$E \to TE'$			$E \to TE'$		
E′		$E' \to \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T′		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

◆思路：

1）把产生式填到何处？

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

一种是：$\alpha \overset{*}{\Rightarrow} a\cdots$

另种是：$\alpha \overset{*}{\Rightarrow} \varepsilon$

什么时候使用ε产生式?

例

- 文法
  - ① $S \rightarrow aBC$
  - 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 $a\ d\ a$　　$a\ d\ e$

- 推导

S	S
$\Rightarrow aBC$	$\Rightarrow aBC$
$\Rightarrow adBC$	$\Rightarrow adBC$
$\Rightarrow adC$	$\Rightarrow adC$
$\Rightarrow ada$	

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）

答疑安排

- 每周二下午2:30-4:30
- 计A321-1

计算 FIRST和 FOLLOW集合

定义　令$G[S]=(V_T,V_N,S,P)$,则对任一文法串α，$\alpha \in (V_T \cup V_N)^*$

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

若$\alpha \overset{*}{\Rightarrow} \varepsilon$，则规定$\varepsilon \in FIRST(\alpha)$

FIRST(α)是α所有可能推导的开头终结符号或可能推导出的ε所构成的集合

如何构造FIRST(α)

$\alpha \in (V_T \cup V_N)^*$

- 单个文法符号的FIRST集合构造方法

  当$\alpha \in (V_T \cup V_N)$ FIRST(α)？

- 在单个文法符号的基础上，任一文法符号串的FIRST集合构造方法

  当$\alpha = X_1X_2\ldots X_n$ FIRST(α)？

单个文法符号的FIRST集合构造方法

当$\alpha \in (V_T \cup V_N)$ FIRST(α)？

a）如果$\alpha \in V_T$，则FIRST(α)={α}

b）如果$\alpha \in V_N$，且有产生式$\alpha \overset{*}{\Rightarrow} a\ldots, a \in V_T$，则把a加入到FIRST($\alpha$)；如果$\alpha \rightarrow \varepsilon$也是一个产生式则把$\varepsilon$加入到FIRST($\alpha$)

任一文法符号串的FIRST集合构造方法

当$\alpha = X_1X_2\ldots X_n$ FIRST(α)？

把 FIRST(X_1)的所有非ε-元素加入到FIRST(α)，若ε在 FIRST(X_1)中，把 FIRST(X_2)的所有非ε-元素也加入到 FIRST(α)，若ε既在 FIRST(X_1)中也在FIRST(X_2)中，把 FIRST(X_3)的所有非ε-元素也加入到 FIRST(α)，…，若对所有的FIRST(X_i) i=1..n,都含有ε，把ε加入到 FIRST(α)。

例子

文法：

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

计算FIRST(E), FIRST(E') , FIRST(T) , FIRST(T') , FIRST(F) ?

FIRST(E)= { (, id }

FIRST(T)= { (, id }

FIRST(F)= { (, id }

FIRST(E')= { + , ε }

FIRST(T')= { * , ε }

FIRST(TE')?

把FIRST(T) 中所有非ε-元素加入到 FIRST(TE')

FOLLOW集定义

文法中任意非终结符FOLLOW(A)定义如下：

令$G[S]=(V_T,V_N,S,P)$，　则

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

若$S \overset{*}{\Rightarrow} \ldots A$，则规定$\#\in Follow(A)$，‘#’作为输入串的右结尾符号。

Follow(A)是所有句型中出现在紧接A之后的终结符号或#所构成的集合

构造每个非终结符的FOLLOW集合

- 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：
  - 对于文法的开始符号S，置 # 于FOLLOW(S)中；
  - 若A→αBβ是一个产生式，则把FIRST(β)-{ε}加至FOLLOW(B)中；
  - 若A→αB是一个产生式，或A→αBβ是一个产生式而$\beta \overset{*}{\Rightarrow} \varepsilon$(即ε∈FIRST(β))，则把FOLLOW(A)加至FOLLOW(B)中

A→αBβ是一个产生式，

∵ ∀a∈ FIRST(β))\{ε}，有$\beta \overset{*}{\Rightarrow}$ a…

则有 $S \overset{*}{\Rightarrow}$ … A …⇒… αBβ … $\overset{*}{\Rightarrow}$ … αBa …

∴a∈ FOLLOW(B)

构造每个非终结符的FOLLOW集合

- 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：
  - 对于文法的开始符号S，置 # 于FOLLOW(S)中；
  - 若A→αBβ是一个产生式，则把FIRST(β)-{ε}加至FOLLOW(B)中；
  - 若A→αB是一个产生式，或A→αBβ是一个产生式而$\beta \overset{*}{\Rightarrow} \varepsilon$(即ε∈FIRST(β))，则把FOLLOW(A)加至FOLLOW(B)中

若A→αB是一个产生式

A→αBβ是一个产生式而$\beta \overset{*}{\Rightarrow} \varepsilon$

注：FOLLOW集合中不能有ε

∴ $S \Rightarrow \ldots Aa \ldots \Rightarrow \ldots \alpha Ba \ldots$

$S \overset{*}{\Rightarrow} \ldots Aa \ldots \Rightarrow \ldots \alpha B\beta a \ldots \overset{*}{\Rightarrow} \ldots \alpha Ba \ldots$

例子

文法：

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

计算Follow(E), Follow(E') , Follow(T) , Follow(T') , Follow(F) ?

Follow(E) = { # ,) }

Follow(E') = { # ,) }

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

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

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

注意：FIRST集针对终结符，非终结符，候选式而构造；

FOLLOW集只对非终结符构造。

例子

文法： *stmt-sequence* →*stmt stmt-seq'*

stmt-seq' → ; *stmt-sequence* | ε

stmt → s

计算 First(*stmt-sequence*), First(*stmt-seq'*) , First(*stmt*)

Follow(*stmt-sequence*) , Follow(*stmt-seq'*) , Follow(*stmt*)

First(***stmt-sequence***) = {s}

First(***stmt***) = {s}

First(***stmt-seq'***) = { ; , ε }

Follow(***stmt-sequence***) ={#}

Follow(***stmt-seq'***) ={#}

Follow(***stmt***)={ ; , # }

非递归预测分析表的构造

在对文法G的每个非终结符号A及其任意候选α都构造出FIRST（α）和FOLLOW（A）之后，可以构造G的分析表M[A，a]。

算法　非递归预测分析表的构造

(1)对文法G的每个产生式$A\to\alpha$执行第2步和第3步；

(2)对每个终结符号$a\in FIRST(\alpha)$，把

$A\to\alpha$加至M[A,a]中；

(3)若$\alpha\overset{*}{\Rightarrow}\varepsilon$，则对任何$b\in FOLLOW(A)$，

把$A\to\alpha$加至M[A,b]中；

WHY

(4)把所有无定义的M[A,a]标上错误标记。

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$SELECT(A\rightarrow\beta)$

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

产生式的可选集

- 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$SELECT(A \rightarrow \alpha) = (FIRST(\alpha) - \{\varepsilon\}) \cup FOLLOW(A)$

文法：　stmt-sequence → stmt stmt-seq'

stmt-seq' → ; stmt-sequence | ε　　stmt → s

stmt-sequence →stmt stmt-seq'

First(stmt stmt-seq') = {s}

stmt-seq' → ; stmt-sequence

First(; stmt-sequence) = {;}

stmt-seq' → ε

Follow(stmt-seq') ={#}

stmt → s

First(s) = {s}

First(stmt-sequence) = {s}

First(stmt) = {s}

First(stmt-seq') = { ; , ε }

Follow(stmt-sequence) ={#}

Follow(stmt-seq') ={#}

Follow(stmt)={ ; , # }

M[A,a]	s	;	#
stmt-sequence	stmt-sequence → stmt stmt-seq'	error	error
stmt	stmt → s	error	error
stmt-seq'	error	stmt-seq' → ; stmt-sequence	stmt-seq' → ε

给出句子s;s;的分析过程

M[A,a]	s	;	#
stmt-sequence	stmt-sequence → stmt stmt-seq'	error	error
stmt	stmt → s	error	error
stmt-seq'	error	stmt-seq' → ; stmt-sequence	stmt-seq' → ε

文法：stmt-sequence → stmt stmt-seq'

stmt-seq' → ; stmt-sequence | ε　　stmt → s

文法：E →TE’　　E’ → +TE’ | ε

T →F T’　　T’ →*F T’ | ε　　F →(E)|id

FIRST(E)= { (, id } FIRST(T)= { (, id } FIRST(F)= { (, id }

FIRST(E’)= { + , ε } FIRST(T’)= { * , ε }

Follow(E) = { # ,) }　Follow(E’) = { # ,) }

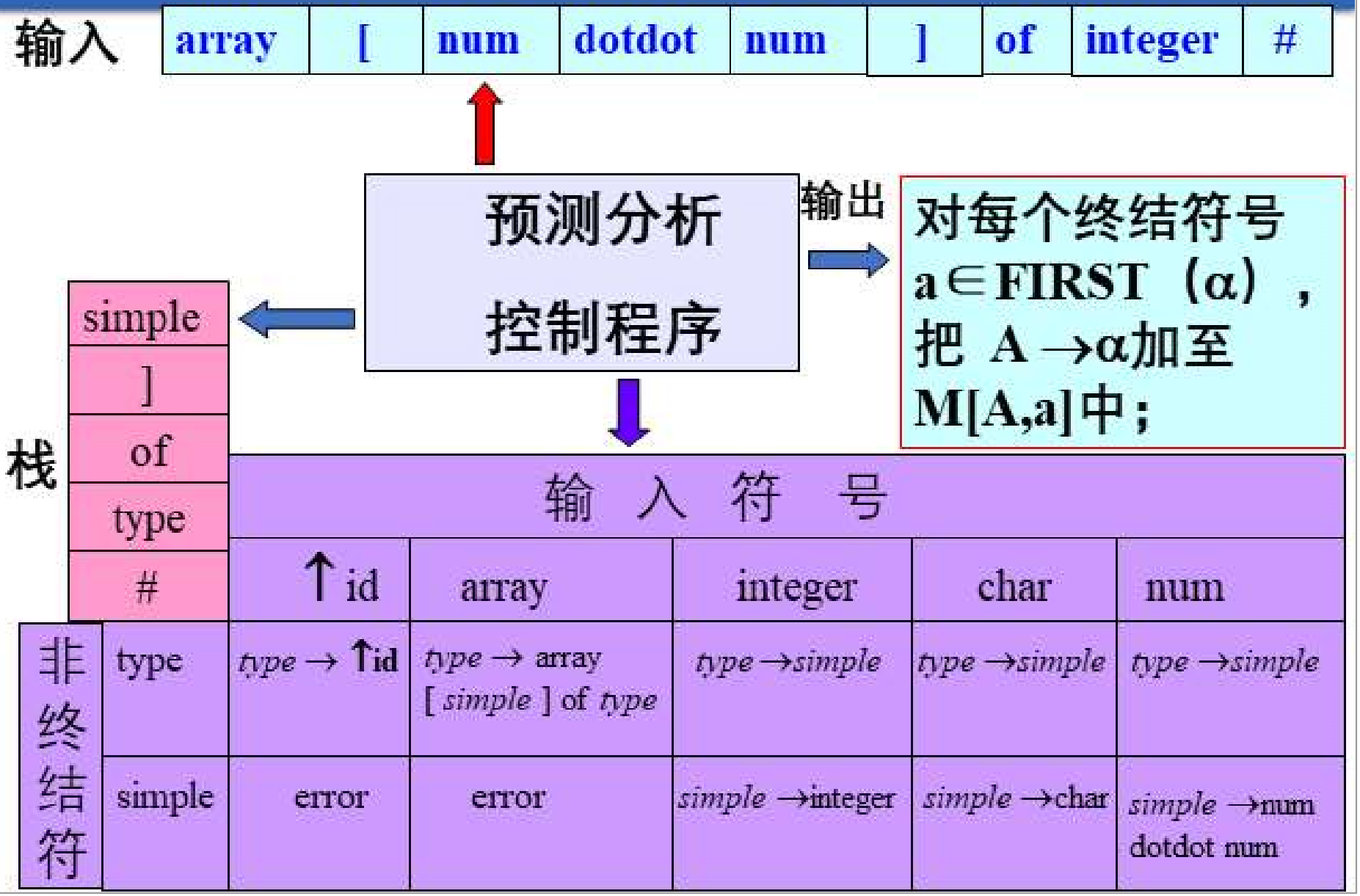
Follow(T) = { # ,) ,+}　Follow(T’) = { # ,) , + }

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

文法分析表M?

非终结符号	输入符号					
	id	+	*	(	)	#
E						
E’						
T						
T’						
F						

思考：构造预测分析表的思想？

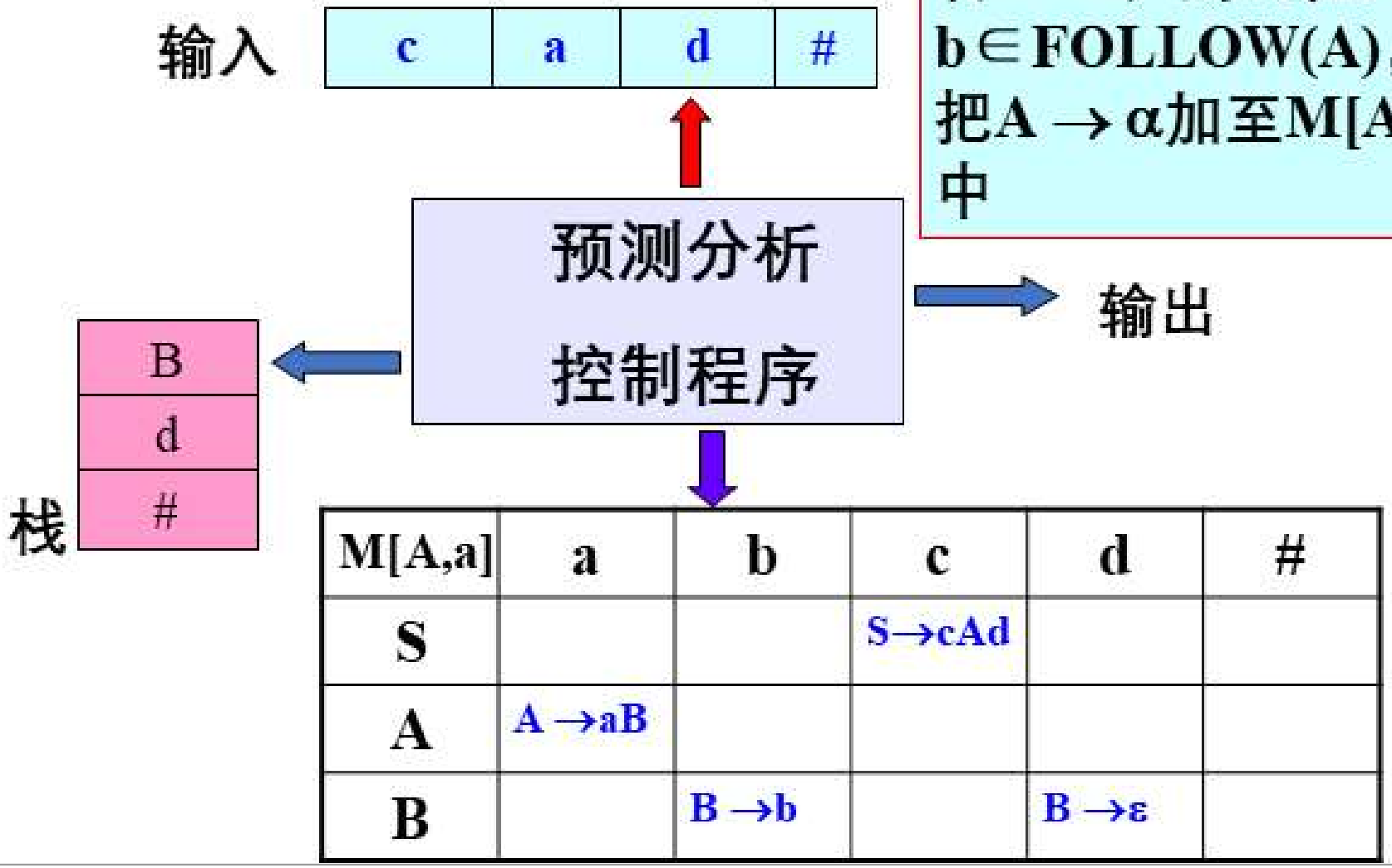


		输入符号				
		↑id	array	integer	char	num
非终结符	type	*type* → ↑id	*type* → array [*simple*] of *type*	*type* →*simple*	*type* →*simple*	*type* →*simple*
	simple	error	error	*simple* →integer	*simple* →char	*simple* →num dotdot num

思考：构造预测分析表的思想？

$S \to cAd \quad A \to aB \quad B \to b \mid \varepsilon$

若$\alpha \overset{*}{\Rightarrow} \varepsilon$，则对任何$b \in FOLLOW(A)$，把$A \to \alpha$加至M[A,b]中



M[A,a]	a	b	c	d	#
S			S→cAd		
A	A →aB				
B		B →b		B →ε	

LL(1)文法

一个文法G，若它的分析表M不含多重定义入口，则被称为LL(1)文法。

可以证明：如果G是左递归的，或者是二义性的文法，则至少有一个多重入口.

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.

一个文法G是LL（1）的，当且仅当对于G的每一个非终结符号A的任何两个不同产生式$A\rightarrow\alpha|\beta$，下面的条件成立：

- FIRST（α）∩FIRST（β）= Φ； α和β推导不出以同一终结符号为首的符号串。
- 假若$\beta \Rightarrow \varepsilon$，那么 FIRST（α）∩ FOLLOW（A）= Φ.

文法：

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

是否LL(1)文法？

条件语句文法不能改造成LL(1)文法

条件语句 → if 条件 then 语句 else 语句
|if 条件 then 语句
|其他

例　　S → iCtS|iCtSeS|a
C → b

提公因子后，文法成为：

S → iCtSS′|a
S′ → eS|ε
C → b

分析步骤

① 计算该文法的FIRST集和FOLLOW集为：

文法：$S \rightarrow iCtSS' | a$

$S' \rightarrow eS | \varepsilon$

$C \rightarrow b$

FIRST(S) = { i， a }

FIRST(S')= {e， ε}

FIRST(C) = { b }

FOLLOW(S) = { #， e }

FOLLOW(S')={ #， e }

FOLLOW(C) = { t }

② 按构造分析表算法，该文法的分析表M为：

文法：S → iCtSS′|a
S′ → eS|ε
C → b

FIRST(S) = { i， a }
FIRST(S’)= {e， ε}
FIRST(C) = { b }
FOLLOW(S) = { #， e }
FOLLOW(S’)={ #， e }
FOLLOW(C) = { t }

	a	b	e	i	t	#
S	$S \rightarrow a$			$S \rightarrow iCtSS'$		
S′			$S' \rightarrow eS$			
C		$C \rightarrow b$				

分析步骤

③候选式 S'→ ε填法(S′ → eS|ε)：

∵ ε∈FIRST(S') = { e， ε}

而： FOLLOW(S')={ #, e }

∴S'->ε填入 M [S', #] 和 M [S', e]，即

	a	b	e	i	t	#
S	$S \rightarrow a$			$S \rightarrow iCtSS'$		
S′			$S' \rightarrow \varepsilon$ $S' \rightarrow eS$			$S' \rightarrow \varepsilon$
C		$C \rightarrow b$				

分析

由上表可见，改造后的文法仍然是非LL(1)文法。（这是因为， M[S', e] 含有多个候选式;
或说：FIRST(eS)∩FOLLOW(S') = { e } ≠φ）
因此，

强制令 M [S',e] = { S' → eS }（即：坚持把e和最近的t相结合。）
从程序语言来看，相当于规定ELSE坚持与最近的THEN相结合。

S' → eS 这个选择就相当于把else和前面最近的then关联起来。如果选S' → ε，将使得else永远不可能被放到栈中或者从输入中被消除，因此选择这个产生式一定是错误的。

例如：if b then s else s

语法分析不确定性的解决办法

- 采用递归分析（回溯）
  - 效率较低
- 改写文法
  - 将非LL(1)文法改写成等价的LL(1)文法
- 无法改写时
  - 增加其他的判别因素
  - 文法过于复杂，无法用自顶向下的方法处理

预测分析法实现步骤

1）构造文法

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

4）检查是不是*LL*(1) 文法。若是，构造预测分析表

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。

递归下降分析程序的再构造

- 根据first集合选择产生式去匹配
- 根据follow集合提前识别错误的句子

递归下降子程序设计

- 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

FOLLOW(E′)={), # }

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

```
PROCEDURE  E′;
IF SYM='+' THEN
BEGIN
        ADVANCE;
        T;  E′
END
ELSE  IF SYM<>'#' AND SYM<>')'
        THEN ERROR
```

递归下降子程序设计

- 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′;
IF SYM=‘*’ THEN
BEGIN
   ADVANCE;
   F;  T′
END;
```

```
PROCEDURE  T′;
IF SYM=‘*’ THEN
BEGIN
        ADVANCE;
        F;  T′
END
ELSE  IF SYM<>‘#’  AND
          SYM<>’)’ AND SYM<>’+’
       THEN  ERROR
```

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

递归下降子程序设计

- 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
- 还可以进一步改造吗？

```
主程序:
PROGRAM PARSER;
BEGIN
   ADVANCE;
   E;
   IF SYM <>'#' THEN
      ERROR
END;
```

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

预测分析中的错误恢复

- 恐慌模式
  - ➢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

例子

非终结符	输入符号					
	id	+	*	(	)	$
E	E→TE′			E→TE′	synch	synch
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′	synch		T→FT′	synch	synch
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id	synch	synch	F→(E)	synch	synch

X	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*，报错，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

例子

非终结符	输入符号					
	id	+	*	(	)	$
E	E→TE'			E→TE'	synch	synch
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'	synch		T→FT'	synch	synch
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id	synch	synch	F→(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'$	$	
$	$	

练习

对文法G(S)：

S → S∨aT | aT | ∨aT

T → ∧aT | ∧a

(1) 消除该文法的左递归和提取左公因子；

(2) 构造各非终结符的FIRST和FOLLOW集合；

(3)构造该文法的LL(1)分析表，并判断该文法是否是LL(1)的

解题

$S \to S \vee a\,T \mid a\,T \mid \vee a\,T$

$T \to \wedge a\,T \mid \wedge a$

(1) 消除左递归，提取左因子

$S \to a\,T\,S' \mid \vee a\,T\,S'$

$S' \to \vee a\,T\,S' \mid \varepsilon$

$T \to \wedge a\,T'$

$T' \to T \mid \varepsilon$

解题

$$S \rightarrow a\,T\,S' \mid \vee a\,T\,S'$$

$$S' \rightarrow \vee a\,T\,S' \mid \varepsilon$$

$$T \rightarrow \wedge a\,T'$$

$$T' \rightarrow T \mid \varepsilon$$

(2) FIRST(S)={a，∨}

FIRST(S')={∨，ε}

FIRST(T)={∧}

FIRST(T')={∧，ε}

FOLLOW(S)={#}

FOLLOW(S')={#}

FOLLOW(T)={∨，#}

FOLLOW(T')={∨，#}

解题

$S \to aTS' \mid \vee aTS'$
$S' \to \vee aTS' \mid \varepsilon$
$T \to \wedge aT'$
$T' \to T \mid \varepsilon$

FIRST(S)={a，∨}　FIRST(S')={∨，ε}
FIRST(T)={∧}　FIRST(T')={∧，ε}
FOLLOW(S)={#}　FOLLOW(S')={#}
FOLLOW(T)={∨，#}　FOLLOW(T')={∨，#}

(3) LL(1)分析表如下，该文法是LL(1)文法。

	a	∨	∧	#
S	S→aTS'	S→∨aTS'		
S'		S'→∨aTS'		S'→ε
T			T→∧aT'	
T'		T'→ε	T'→T	T'→ε

例子

已知文法G=({a,b,c},{S,A,B},S,P)

其中P:

S→aBc|bAB

A→aAb|b

B→b|ε

（1）构造该文法的LL(1)分析表。

（2）给出句子baabbb的分析过程。

(分析步骤格式: 栈 输入缓冲区 动作)

例子

已知文法G=({a,b,c},{S,A,B},S,P)

其中P:

S→Sc|bAB

A→aAb|b

B→b|ε

构造该文法的LL(1)分析表。