北京大学信息科学技术学院

2022年春季学期《编译原理》

第 4 章 语法分析（2）

Syntax Analysis

【对应教材 2.4, 4.3, 4.4】

回顾

- 上下文无关文法

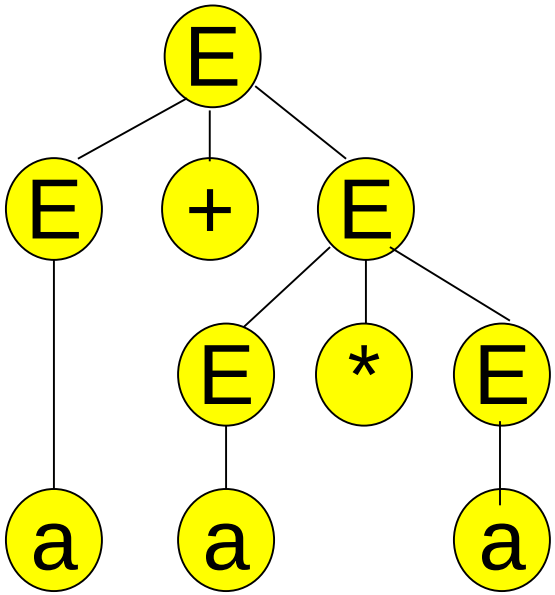
  $E \rightarrow E+E \mid E*E \mid (E) \mid a$

- 推导

  $E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E*E \Rightarrow a+a*E \Rightarrow a+a*a$

  - 最左推导、最右推导

- 分析树



- 二义性

内容提要

- 语法分析简介
- 上下文无关文法
- 文法的设计方法
- **自顶向下的语法分析**
- 自底向上的语法分析
  - 简单LR分析：LR(0), SLR
  - 更强大的LR分析：LR(1), LALR
  - 二义性文法的使用

自顶向下的语法分析

- 自顶向下分析是从文法的开始符号出发，试构造出一个最左推导，从左至右匹配输入的单词串。
- 如果在每步推导中，将要被替换的非终结符号是A、而当前从左至右读入输入串读到的单词符号是a，如果A为左部的所有产生式（假定无ε-产生式）是：

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

并且只有α_i $(1 \le i \le n)$推导出的第一个终结符号是a，那么，就可以用产生式$A \rightarrow \alpha_i$ 构造最左推导，即用α_i替换A。则可作预测分析；否则只能带回溯地进行尝试。

自顶向下分析方法的特点

- 分析过程是带预测的
  - 对输入符号串要预测属于什么语法成分，然后根据该语法成分的文法建立语法树。
- 分析过程是一种试探过程
  - 选用不同规则来建立语法树
  - 由于是试探过程，难免会有失败，所以分析过程需要进行回溯，因此也称这种方法是带回溯的自顶向下分析方法。

- 然而，带回溯的自顶向下分析方法在实际应用中价值不大，效率很低。

递归下降的语法分析

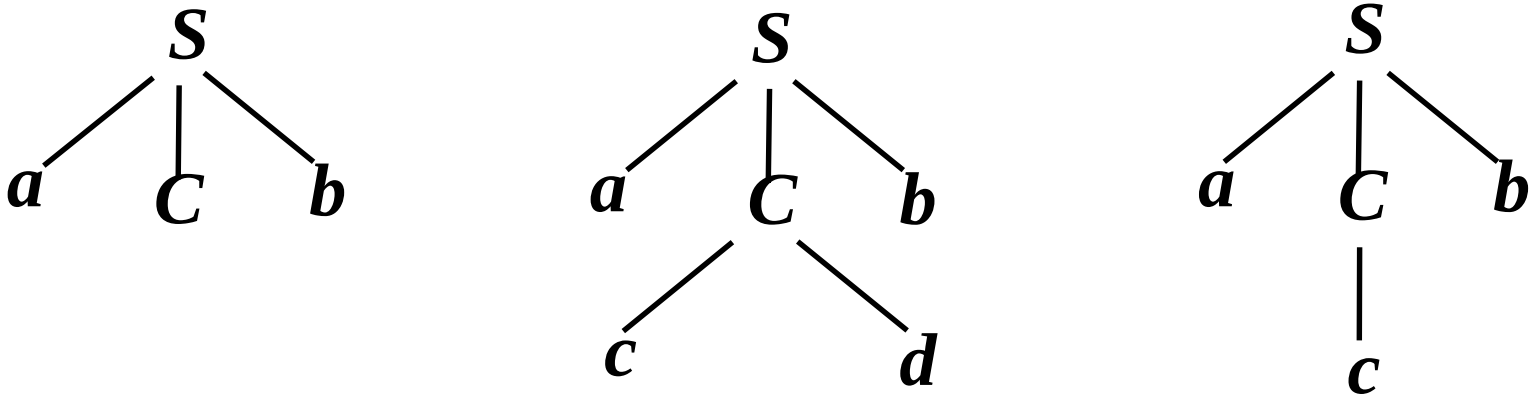
- 递归下降语法分析程序由一组过程组成
- 每个非终结符号对应于一个过程，该过程负责扫描非终结符号对应的结构
- 程序执行从开始符号对应的过程开始
  - 当扫描整个输入串时宣布分析成功完成

```
    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 */;
        }
    }
```

回溯的例子

例：文法 $S \to aCb \quad C \to cd \mid c$

为输入串 $w = acb$ **建立分析树：**



递归下降分析技术的回溯

- 前面的算法报告错误（第7行）并不意味着输入串不是句子，而可能是表示前面选错了产生式。
  - 第1行上保存当前的扫描指针
  - 在第7行上应该改成
    - 回退到保存的指针；
    - GOTO (1) 并选择下一个产生式；
    - 如果没有产生式可选，报告错误。

```
    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 */;
        }
    }
```

- 回溯需要来回扫描，低效
- 解决方法：设法通过一些信息确定唯一可能的产生式

如何保证没有回溯？

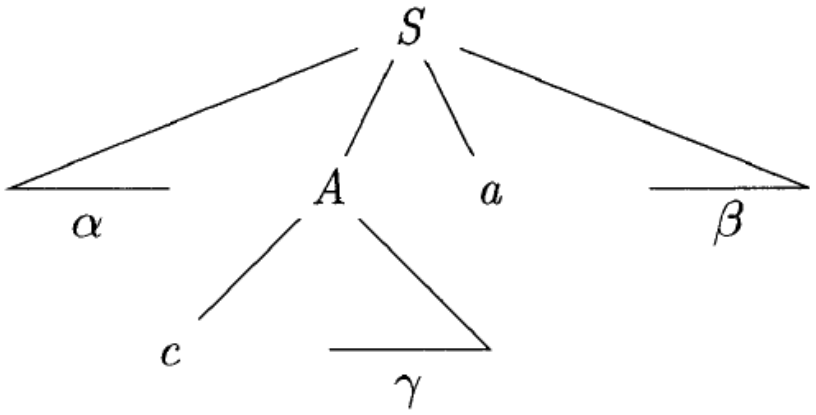
- 对文法加什么样的限制可以保证没有回溯？
- 在自顶向下的分析技术中，通常使用向前看几个符号来唯一地确定产生式（这里假定只看一个符号）
- 假设当前句型是**xAβ**，而输入是**xa…**。那么选择产生式**A → α**的必要条件是下列之一：
  - **α =>* ε**，且β以a开头；也就是说在某个句型中**a跟在A之后**；
  - **α =>* a…**；
  - 如果按照这两个条件选择时能够保证唯一性，那么我们就可以避免回溯。

First 和 Follow

- First (α)
  - 可以从α推导得到的串的首符号的集合
  - First (α)={ $a \mid \alpha \Rightarrow^* a\ldots, a \in V_T$}
    特别的，$\alpha \Rightarrow^*$ ε时，规定ε ∈ First (α)
- Follow (A)
  - 可能在某些句型中紧跟在A右边的终结符号的集合
  - Follow (A) = {$a \mid S \Rightarrow^* \ldots Aa\ldots$, $a \in V_T$}
    如果A是某个句型的最右符号时，那么$属于Follow(A)

右图中:

- c∈First(A),
- α∈Follow(A)



First集合的计算方法(1)

- 先计算单个符号X的First集合
  - 如果X是终结符号，那么First (X)=X
  - 如果X是非终结符号，且$X \rightarrow Y_1 Y_2 \ldots Y_k$是产生式
    - 如果a在First(Y_i)中，且ε在First(Y_1), First(Y_2),…, First(Y_{i-1})中，那么a也在First(X)中。
    - 如果ε在First(Y_1), First(Y_2),…, First(Y_k)中，那么ε在First(X)中。
  - 如果X是非终结符号，且$X \rightarrow \varepsilon$是一个产生式，那么ε在First(X)中

First集合的计算方法(2)

- 再计算产生式右部$X_1X_2…X_n$的First集合
  - 向集合中加入First(X_1)中所有非ε的符号；
  - 如果ε在First(X_1)中，再加入First(X_2)中的所有非ε的符号；
  - …
  - 如果ε在所有的First(X_i)中，将ε加入First($X_1X_2…X_n$)中。

Follow集合的计算方法

- 算法
  - 将右端结束标记$放到Follow(S)中
  - 按照下面的两个规则不断迭代，直到所有的Follow集合都不再增长为止。
    - 如果存在产生式A → αBβ，那么First(β)中所有非ε的符号都在Follow(B)中。
    - 如果存在一个产生式A → αB，或者A → αBβ且First(β)包含ε，那么Follow(A)中的所有符号都加入到Follow(B)中。

Warning：请注意各个步骤中将符号加入到Follow集合中的理由。

例：把上述算法应用于文法G：

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

$T \rightarrow FT'$　　$T' \rightarrow *FT' | \varepsilon$　　$F \rightarrow id \mid (E)$

First(F) = {(, id};

First(E) =First(T) = {(,id}

First(E') = {+, ε};　　First(T')={*, ε}

First(*TE'*) = First(*FT'*) = { (, id}　　First(ε)={ε}

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

First((E)) = {(}　　First(id) = {id}

Follow(E) = Follow (*E'*) = {), $}

Follow(*T*) = Follow (*T'*) = { +,), $}

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

LL(1)文法

- **LL(1)文法的定义：**

  对于文法中任意两个不同的产生式A→α|β

  1. **不存在终结符号a使得α和β都可以推导出以a开头的串**
  2. **α和β最多只有一个可以推导出空串**
  3. **如果β可以推导出空串，那么α不能推导出以Follow中任何终结符号开头的串；**

- **等价于：**
  - **First(α)∩First(β)=Φ；（条件1、2）**
  - **如果ε∈First(β)，那么First (α) ∩Follow(A)=Φ；反之亦然。（条件3）**

LL(1)文法的说明

- LL(1)中的第一个“L”表示从左到右扫描输入;
- 第二个“L”表示生成一个最左推导;
- “1”表示为做出分析动作的决定，每一步只需向前看 1 个符号。
- 类似可以定义LL(k) 文法。
- 输入串以$为结束标记。这相当于对文法作扩充，即增加产生式 S’→S$。所以Follow(S)一定包含$。

预测分析表的构造方法

- 输入：文法G
- 输出：预测分析表M
- 方法：
  - 对于文法G的每个产生式A→α
    - 对于First(α)中的每个终结符号a，将A→α加入到M[A,a]中；
    - 如果ε在First(α)，那么对于Follow(A)中的每个符号b，将A→α加入到M[A,b]中。
  - 最后在所有的空白条目中填入error。

预测分析表的例子

- 文法：
  - E→TE’　　E’→+TE’ | ε
  - T → FT’　　T’→*FT’ | ε　　F→(E) | id
- FIRST集：
  - F：{(, id}；　　E,T：{(,id}；　　E’：{+, ε}；　　T’：{*, ε}
- FOLLOW集：
  - E：{$,)}　　E’：{$,)}　　T,T’：{+,), $}　　F：{+,*,),$}

非终结符号	输入符号					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

（非回溯的）预测分析法

- 采用预测分析法要求文法是LL(1)文法

- 递归下降子程序方法
- 表驱动的（非递归）预测分析法

递归下降子程序方法

- 递归下降语法分析程序由一组过程组成
- 每个非终结符号对应于一个过程，该过程负责扫描非终结符号对应的结构
- 可以使用当前的输入符号来唯一地选择产生式

```
    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 */;
        }
    }
```

如果当前输入符号为a，那么选择M[A,a]中的产生式

分析表达式的递归下降子程序(1)

```
void f_e ( );
    {
        f_t( );  f_e'( );
    }

void  f_e' ( );
   {
      if  (lookahead == ′ + ′ )
          { match(′ + ′ ); f_t() ; f_e'();  }
      else {
          if  ( lookahead != ′ ) ′ && lookahead != ′$′ )
                    { error ();  }
       }
    };
```

$E \rightarrow TE'$

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

分析表达式的递归下降子程序(2)

$T \rightarrow FT'$

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

```
void  f_t()
    {
        f_f();  f_t'();
    }

void  f_t' ()
   {
      if  ( lookahead == ' * '  )
          { match(' * ' );  f_f();  f_t'(); }
      else {
         if ( lookahead != ' + '  && lookahead != ' ) '
                              &&  lookahead != '$'  )
                  { error();     }
        }
    }
```

分析表达式的递归下降子程序(3)

$F \rightarrow id \mid (E)$

```
void  f_f ()
    {
       if  ( lookahead== '(' )
           {
              match('('); f_e() ; match(') ') ;
            }
       else {  if  ( lookahead == id)
                       { match(id); }
                  else  { error(); }
           }
     }
```

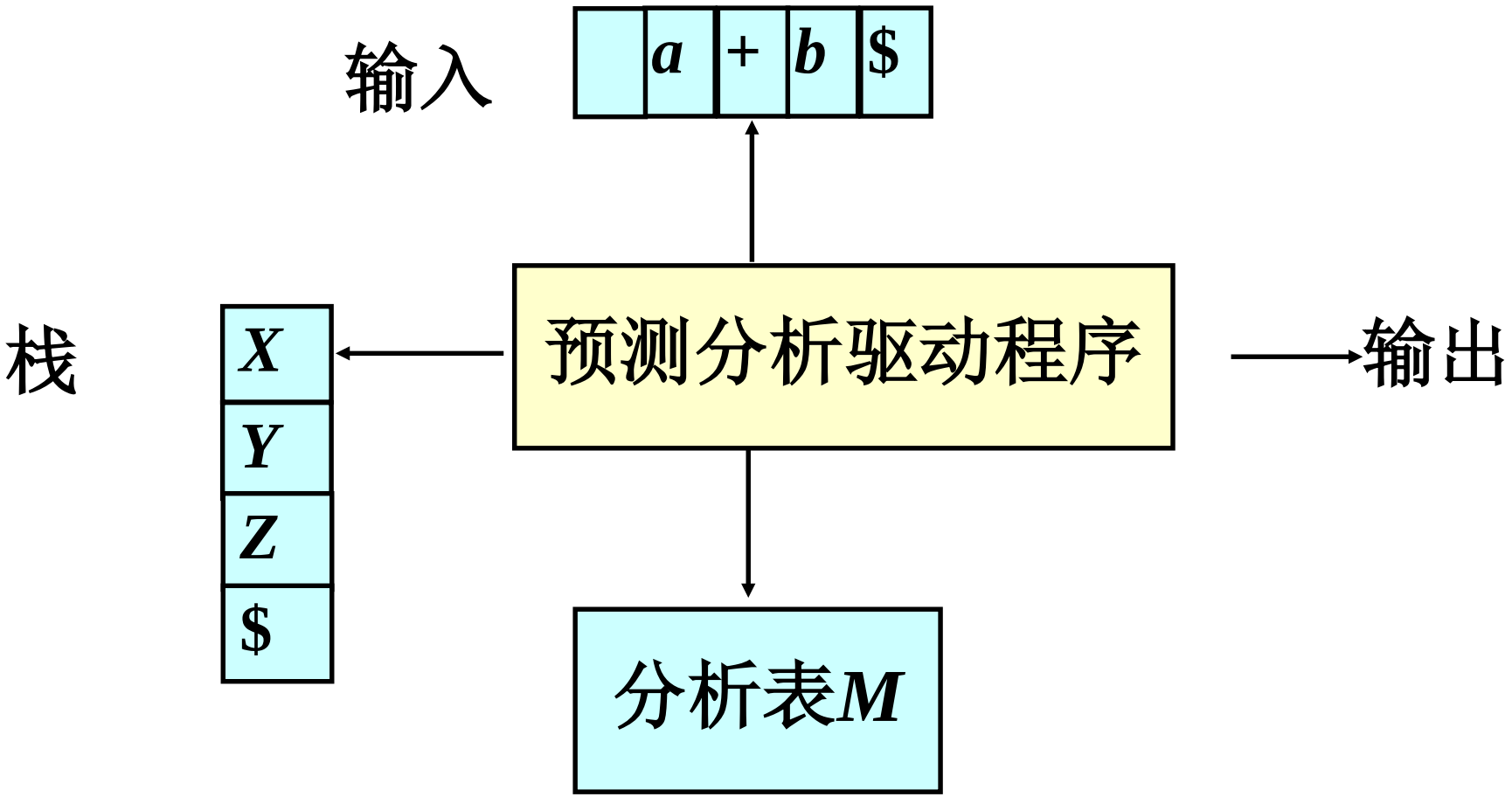
表驱动的(非递归)预测分析

- 需要
  - 一个预测分析驱动程序
  - 一个预测分析栈
  - 一个预测分析表

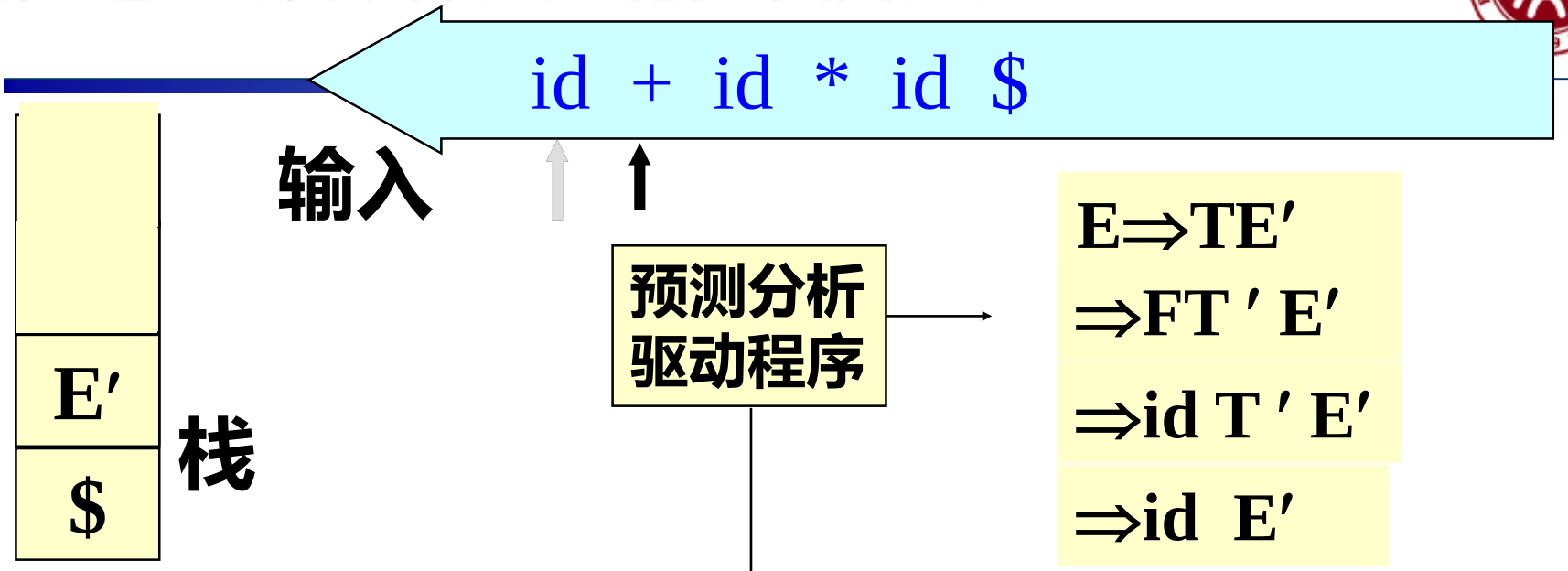
- 前两项与具体文法无关，对所有文法都是相同的；

- 预测分析表依赖于具体的LL(1)文法。

非递归的预测分析



非递归预测分析器的模型



非终结符号	输入符号					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

非递归预测分析器的驱动程序

把 ip 指向 w$ 的第一个符号；令x为栈顶符号，a是 ip 所指向的符号。

```
push(st,$); push(st,S);
REPEAT
   IF  x∈V_T∪ {$}
   THEN  IF  x=a  THEN { pop(st);  ip:=ip+1}
                ELSE  error( )
   ELSE   IF  M[x,a] = x→y1y2…yk
        THEN { pop(st); push(st,yk);…;push(st,y1);
                    write('x→y1y2…yk')  }
        ELSE   error( )
UNTIL   x=$  /*栈为空*/
```

非LL(1)文法

- 二义性文法肯定不是LL(1)文法，例如：

  G[S]: S→iEtSS′ | a　　S′→ eS | ε　　E→b

- Follow(S′)={e,$}
- 其相应的分析表如下表

	a	b	e	i	t	$
S	S→a			S→iEtSS′		
S'			S′ →eS S′ →ε			S′→ε
E		E→b				

非LL(1)文法

- 没有提取左公因子的文法不是LL(1)文法
- 如G的产生式为：

  $S \rightarrow aSb \mid ab \mid a$

  对任意k，语言L(G) 没有相应的LL(k)文法。

- 左递归文法不是LL(1)文法
  - 表达式文法：$E \rightarrow E+T \mid T$
  - First(E+T) = {id,(}
  - First(T) = {id,(}

语法错误的处理

- 程序中可能存在不同层次的错误
  - 词法错误：例如标识符、关键字或运算符拼写错误
  - 语法错误：例如分号位置错误、{}多余或缺失等
  - 语义错误：例如运算符和运算分量类型不匹配
  - 逻辑错误：例如程序的错误推理引起的任何错误。
- 语法分析错误处理的目标
  - 清晰准确报告错误
  - 错误恢复的能力：继续检测下面的错误

预测分析中的错误恢复

- 错误恢复
  - 当预测分析器报错时，表示输入的串不是句子。
  - 用户希望预测分析器能够进行恢复，并继续语法分析过程，在一次分析中找到更多的语法错误。
    - 有可能恢复得并不成功，之后找到的语法错误有可能是假的。
  - 进行错误恢复时可用的信息：栈里面的符号，待分析的符号
- 两类错误恢复方法：
  - 恐慌模式
  - 短语层次的恢复

作业

- 文法变换
  - Ex. 4.3.1, 4.3.3
- 自顶向下分析
  - Ex. 4.4.1 (1, 3, 5小题)
  - (同时计算First和Follow集合）