

## 第四章 语法分析



# 学习重点

---

- 上下文无关文法
- 自顶向下分析方法：递归实现、表驱动
- 自底向上分析方法
  - 算符优先分析方法
  - LR分析方法
    - SLR
    - 规范LR
    - LALR

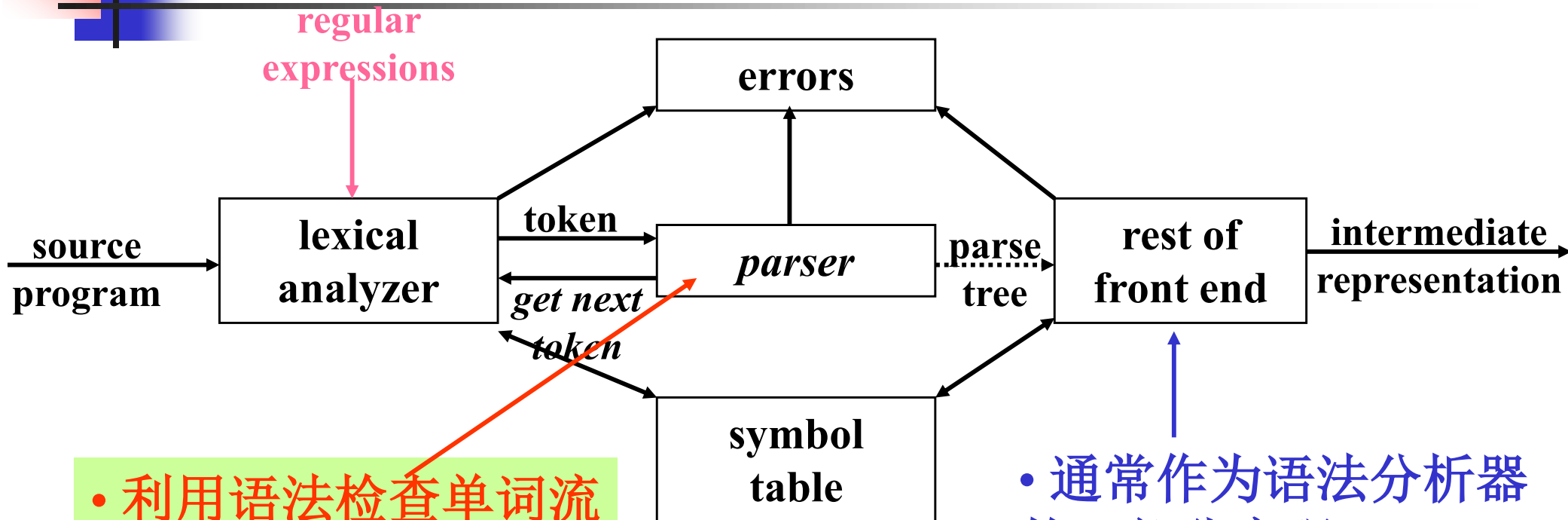


# 概述

---

- 为什么使用上下文无关文法？
  - 精确、容易理解的语法描述
  - 特定类别语法——编译器自动构造  
额外好处——发现二义性和难于分析的结构
  - 加入特殊结构——翻译、错误检测  
基于语法的翻译方法描述 **工具** → 程序
  - 语言发展变化（新语句，语句变化...）  
——容易增加新结构，修改已有部分  
**ADA → ADA9x, C++增加: 模板、异常**

## 4.1 语法分析器的角色



- 利用语法检查单词流的语法结构
- 构造语法分析树
- 语法错误和修正
- 识别正确语法
- 报告错误

- 通常作为语法分析器的一部分实现
- 包括对单词扩充信息, 以进行类型检查、语义分析等工作



# 三类语法分析器

---

- 通用分析器，Cocke-Younger-Kasami算法，适用任何文法，效率低
- 自顶向下分析器，top-down
- 自底向上分析器，bottom-up
- top-down，bottom-up：适用特定类别文法——LL、LR，描述能力足够



## 4.1.1 语法错误处理

---

- 不同层次的错误
  - 词法：拼写错误
  - 语法：单词漏掉、顺序错误
  - 语义：类型错误
  - 逻辑：无限循环/递归调用
- 语法错误处理为重点
  - 语法错误相对较多
  - 编译器容易高效检测



# 错误处理目标

---

- 三个“简单”的目标
  - 清楚、准确地检测、报告错误及其发生位置
  - 快速恢复，继续编译，以便发现后续错误
  - 不能对“正确”程序的编译速度造成很大影响
- 完全实现困难
- LL, LR, 可最快速度发现错误
  - 活前缀特性, **viable-prefix property**
  - 一个输入前缀不是语言中任何符号串前缀——发生错误



## 例4.1

---

### ○ 有关程序错误的统计

- 60%的程序无语法、语义错误
- 错误发生是分散的：错误语句中80%只有一个错误，13%有两个
- 多数错误是简单的：90%的错误是单个单词错误
- 错误分类：60%是标点符号错误，20%是运算符和运算对象错误，15%是关键字错误





## 例4.1（续）

```
#include<stdio.h>
int f1(int v)
{   int i,j=0;
    for (i=1;i<5;i++)
        {   j=v+f2(i) }
    return j;
}
int f2(int u)
{   int j;
    j=u+f1(u*u);
    return j;
}
int main()
{   int i,j=0;
    for (i=1;i<10;i++)
        {   j=j+i*i   printf("%d\n",i);        }
    printf("%d\n",f1(j));
    return 0;
}
```

哪些“容易”恢复？  
哪些“困难”？



## 4.1.2 错误恢复策略

### 1. Panic模式

- 丢弃单词，直到发现“同步”单词
- 设计者指定同步单词集，{end, “;”, “}”, ...}
- 缺点
  - 丢弃输入⇒遗漏定义，造成更多错误
  - 遗漏错误
- 优点
  - 简单⇒适合每个语句一个错误的情况



# 错误恢复策略（续）

---

## 2. 短语级（**phrase level**）

- 局部修正，继续分析
- “,” $\Rightarrow$ “;”，删除“,”，插入“;”
- 同样由设计者指定修正方法
- 避免无限循环
- 有些情况不适用
- 与Panic模式相结合，避免丢弃过多单词



# 错误恢复策略（续）

---

## 3. 错误产生式（error production）

- 理解、描述错误模式
- 文法添加生成错误语句的产生式
- 拓广文法→语法分析器程序
- 如，对C语言赋值语句，为“:=”添加规则  
报告错误，但继续编译
- 错误检测信息+自动修正



# 错误恢复策略（续）

## 4. 全局修正（global correction）

- ❑ 错误程序 → 正确程序
- ❑ 寻找最少修正步骤，插入、删除、替换
- ❑ 不正确输入 $x$ ，文法 $G$   $\xrightarrow{\text{最少修正 } x \rightarrow y}$   $y$ 对应的语法分析树
- ❑ 过于复杂，时空效率低



## 4.2 上下文无关文法

- 定义：四元式( $V_T, V_N, S, \mathcal{P}$ )
  1.  $V_T$ : 终结符号（单词）集,  $T$
  2.  $V_N$ : 非终结符（语法变量）集,  $NT$ , 定义了文法/语言可生成的符号串集合
  3.  $S$ :  $S \in NT$ , 开始符号, 定义语言的所有符号串
  4.  $\mathcal{P}$ , 产生式集,  $PR, NT \rightarrow (T \mid NT)^*$   
规则 $\rightarrow T$ 、 $NT$ 如何组合, 生成语言的合法符号串



## 例4.2：简单表达式

$expr \rightarrow expr \ op \ expr$

$expr \rightarrow ( \ expr )$

$expr \rightarrow - \ expr$

$expr \rightarrow id$

$op \rightarrow +$

$op \rightarrow -$

$op \rightarrow *$

$op \rightarrow /$

$op \rightarrow \uparrow$

蓝色符号——T，黑色符号——NT

优先级不能区为高低

二义性



## 4.2.1 符号约定

---

### 1. 终结符

- 字母表靠前的小写字母, a、b、c ...
- 运算符, +、- ...
- 标点符号, (、)、, ...
- 数字, 0、1、...、9
- 粗体字符串, **id**、**if** ...  
蓝色字符串, id、if ...



# 符号约定 (续)

## 2. 非终结符

- 字母表靠前的大写字母, A、B、C ...
- S, 通常作为开始符号
- 斜体小写字母串, *expr*、*term*、...

## 3. 字母表靠后的大写字母 T/NT 都可以

- X、Y、Z, 语法符号——T或NT

## 4. 字母表靠后的小写字母

- u、v ..., 终结符号串,  $T^*$

$uvw \rightarrow T^*$  符号串



## 符号约定（续）

### 5. 小写希腊字母

□  $\alpha$ 、 $\beta$ 、 $\gamma$ ..., 语法符号串,  $(T \cup NT)^*$

### 6. 左部相同的产生式可合并, ‘|’——“或”

□  $A \rightarrow \alpha_1; A \rightarrow \alpha_2; \dots; A \rightarrow \alpha_k;$   
 $\Rightarrow A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$ , 候选式

### 7. 第一个产生式的左部为开始符号



## 例4.3 利用符号约定简化文法

---

- 例4.2中表达式文法简化后结果

$$E \rightarrow E A E \mid ( E ) \mid -E \mid \mathbf{id}$$
$$A \rightarrow + \mid - \mid * \mid / \mid \uparrow$$



## 4.2.2 推导 (derivation)

---

- 描述文法定义语言的过程
- 自顶向下构造语法分析树的精确描述
- 将产生式用作重写规则
  - 由开始符号起始
  - 每个步骤将符号串转换为另一个符号串
  - 转换规则：利用某个产生式，将符号串中出现的其左部NT替换为其右部符号串

## 推导 (续)

○  $E \rightarrow E + E \mid E * E \mid ( E ) \mid -E \mid \mathbf{id}$

○  $E \rightarrow -E$ ,  $E$ 可替换为 $-E$

$E \Rightarrow -E$ , “ $E$ 直接推出 $-E$ ”

$E * E \Rightarrow (E) * E$

○  $E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(\mathbf{id})$

替换序列,  $E \rightarrow -(\mathbf{id})$ 的一个**推导**

双横线箭头, 一步推导.

# 定义

推导

## ○ 形式化定义

□  $\alpha A \beta \Rightarrow \alpha \gamma \beta$  仅当存在产生式  $A \rightarrow \gamma$

□  $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$  ——  $\alpha_1 \stackrel{*}{\Rightarrow} \alpha_n$

□ 若  $\alpha \stackrel{*}{\Rightarrow} \beta$  且  $\beta \rightarrow \gamma$ , 则  $\alpha \stackrel{*}{\Rightarrow} \gamma$

△ ○  $\Rightarrow$ , “一步推导”, “直接推出”, 推导步数 = 1

$\stackrel{*}{\Rightarrow}$ , “一步或多步推导”, 推导步数  $\geq 1$

$\stackrel{*}{\Rightarrow}$ , “0步或多步推导”, 推导步数  $\geq 0$

# 推导与语言的关系

- 文法G，开始符号S，生成的语言L(G)  
终结符号串w

$$w \in L(G) \Leftrightarrow S \Rightarrow^+ w$$

- w: G的一个**句子**, sentence
- CFG生成**上下文无关语言**
- 两个CFG生成相同语言，两个CFG**等价**
- $S \Rightarrow^* \alpha$ ,  $\alpha$ 可能包含NT  
 $\alpha$ : G的一个**句型**, sentential form  
句子: 不包含NT的句型

句型 包括NT

句子: 不包括NT的句型

## 例4.4

$E \Rightarrow E * E \Rightarrow \mathbf{id} * E \Rightarrow \mathbf{id} * \mathbf{id}$

句型

句子

$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$

另一种推导过程

$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(E+\mathbf{id}) \Rightarrow -(\mathbf{id}+\mathbf{id})$



# 最左推导和最右推导

- **最左推导**: 总替换最左边的NT

$$E \xRightarrow{\text{lm}} -E \xRightarrow{\text{lm}} -(E) \xRightarrow{\text{lm}} -(E+E) \xRightarrow{\text{lm}} -(\text{id}+E) \xRightarrow{\text{lm}} -(\text{id}+\text{id})$$

~ 表示最左边NT替换

- **最右推导**: 总替换最右边的NT

$$E \xRightarrow{\text{rm}} -E \xRightarrow{\text{rm}} -(E) \xRightarrow{\text{rm}} -(E+E) \xRightarrow{\text{rm}} -(E+\text{id}) \xRightarrow{\text{rm}} -(\text{id}+\text{id})$$

w 表示终结符号串

- 形式化定义:  $A \rightarrow \delta$

$$wA\gamma \xRightarrow{\text{lm}} w\delta\gamma, \quad w \text{ 只含 } T$$

$$\beta Aw \xRightarrow{\text{rm}} \beta\delta w, \quad w \text{ 只含 } T$$

最右边NT

- $S \xRightarrow{\text{lm}}^* \alpha$ ,  $\alpha$ : **最左句型**, left-sentential form

最左推导中出现的串



## 4.2.3 语法分析树和推导

---

- 语法树：推导的图示，但不体现推导过程的顺序
  - 内部节点：非终结符A
  - 内部结点A的孩子节点：左 $\rightarrow$ 右，对应推导过程中替换A的右部符号串的每个符号
  - 叶：由左至右 $\rightarrow$ 句型，**yield**, **frontier**



# 语法树与推导的关系

---

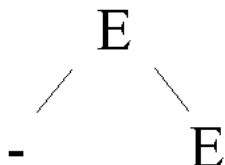
- 一个推导过程:  $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$ 
  - $\alpha_1 \equiv A$ , 单节点, 标记为A
  - $\alpha_{i-1} = X_1 X_2 \dots X_k$  对应语法树T
  - 第i步推导,  $X_j \rightarrow Y_1 Y_2 \dots Y_r$
  - T的第j个叶节点, 添加r个孩子节点 $Y_1, Y_2, \dots, Y_r$ , 特殊情况,  $r=0$ , 一个孩子 $\varepsilon$

## 例4.5

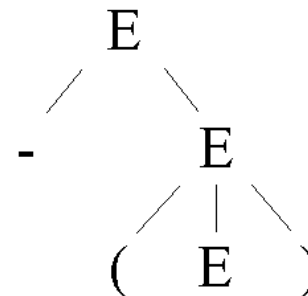
$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$

$E$

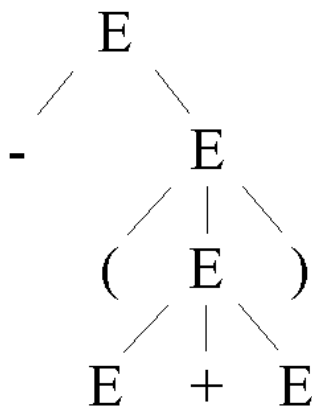
$\Rightarrow$



$\Rightarrow$

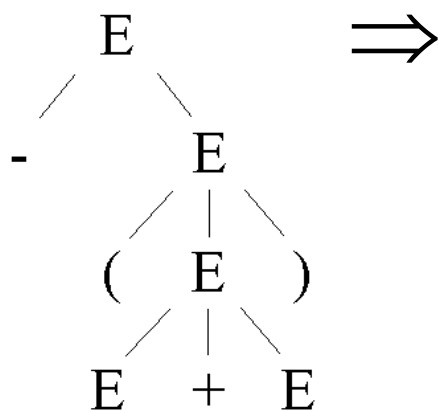


$\Rightarrow$

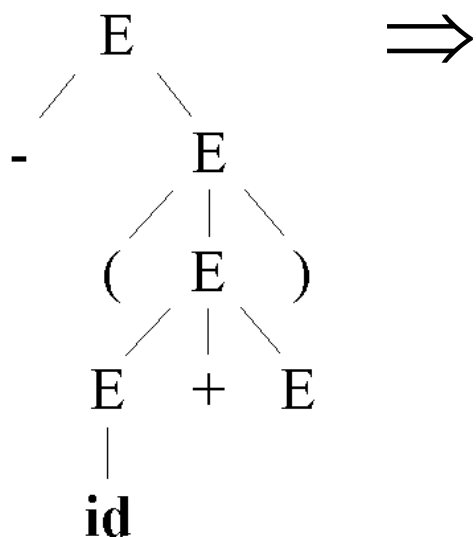


## 例4.5

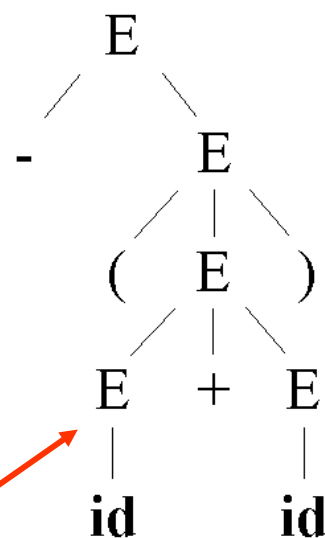
$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$



$\Rightarrow$



$\Rightarrow$



$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(E+\mathbf{id}) \Rightarrow -(\mathbf{id}+\mathbf{id})$

一棵语法树可 $\leftrightarrow$ 多个推导

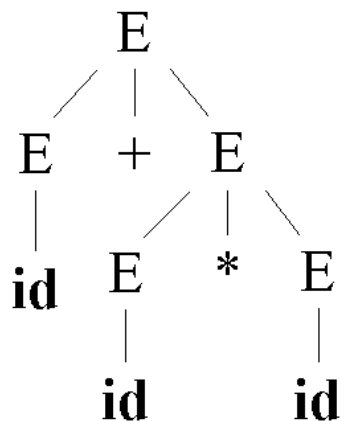
一棵语法树 $\leftrightarrow$ 唯一最左推导，唯一最右推导

↓ 无二义性文法

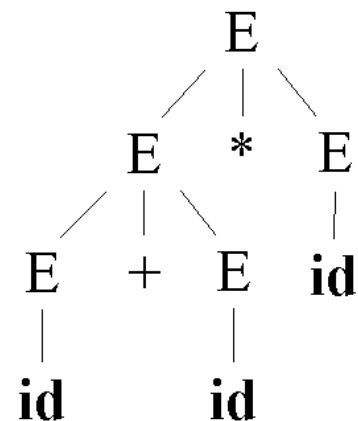
义性的。我们也可以如下定义二义性文法：如果  $L(G)$  中存在一个具有两个或两个以上最左（或最右）推导的句子，则  $G$  是二义性文法。很多语法分析器要求所处理的文法是无二义的，

## 4.2.4 二义性文法（例4.6）

$E \Rightarrow E+E \Rightarrow \mathbf{id}+E$   
 $\Rightarrow \mathbf{id}+E^*E$   
 $\Rightarrow \mathbf{id}+\mathbf{id}^*E$   
 $\Rightarrow \mathbf{id}+\mathbf{id}^*\mathbf{id}$



$E \Rightarrow E^*E \Rightarrow E+E^*E$   
 $\Rightarrow \mathbf{id}+E^*E$   
 $\Rightarrow \mathbf{id}+\mathbf{id}^*E$   
 $\Rightarrow \mathbf{id}+\mathbf{id}^*\mathbf{id}$



○ 句子 $\leftrightarrow$ 多个语法树, 多个最左（右）推导

有多个最左推导/最右推导

↓ 表示有二义性



## 4.3 设计CFG

---

### ○ 4.3.1 正规式与CFG

#### ○ 正规式

- 词法分析的基础
- 描述正规语言
- 描述能力不够,  $a^n b^n, n \geq 1$

#### ○ 上下文无关文法

- 语法分析的基础
- 描述程序语言结构
- 上下文无关语言

# 正规式与上下文无关文法

- 正规式可描述的语言CFG均可描述,

$(a|b)^*abb$

$A_0 \rightarrow aA_0 \mid aA_1 \mid bA_0$

$A_1 \rightarrow bA_2$

$A_2 \rightarrow bA_3$

$A_3 \rightarrow \varepsilon$

Reg. Lang.

CFLs

- 正规语言  $\subset$  上下文无关语言

不确定的有穷自动机（简称为NFA）是一个由以下几部分组成的数学模型：

1. 一个状态的有穷集合  $S$ 。
2. 一个输入符号集合  $\Sigma$ ，即输入符号字母表。
3. 一个转换函数 *move*，它把由状态和符号组成的二元组映射到状态集合。
4. 状态  $s_0$  是惟一的开始或初始状态。
5. 状态集合  $F$  是接受（或终止）状态集合。

NFA

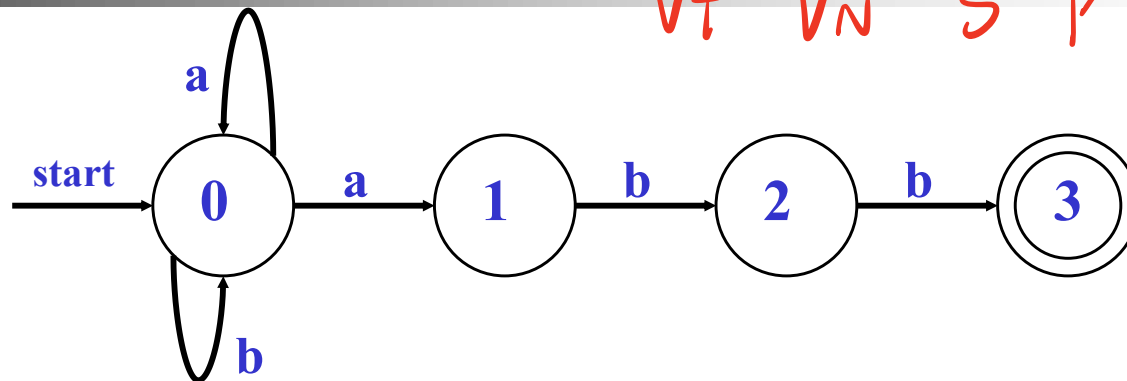
定义回顾

注意



# NFA $\rightarrow$ CFG

$\Sigma$   $S$   $S_0$   $F$   $\delta$   
 $\downarrow$   $\downarrow$   $\downarrow$   
 $V_T$   $V_N$   $S$   $P$



1. 状态  $i \rightarrow$  非终结符  $A_i$ :  $A_0, A_1, A_2, A_3$

2.  $i \xrightarrow{a} j \rightarrow A_i \rightarrow aA_j$

3.  $i \xrightarrow{\epsilon} j \rightarrow A_i \rightarrow A_j$

$\left\{ \begin{array}{l} : A_0 \rightarrow aA_0, A_0 \rightarrow aA_1 \\ : A_0 \rightarrow bA_0, A_1 \rightarrow bA_2 \\ : A_2 \rightarrow bA_3 \end{array} \right.$

4. 若  $i$  为终态  $\rightarrow A_i \rightarrow \epsilon$ :  $A_3 \rightarrow \epsilon$

5. 若  $i$  为初态,  $A_i$  为开始符号:  $A_0$

○ 正则文法

# NFA $\rightarrow$ CFG

① ○  $A_i$  的含义是什么？状态  $i \rightarrow$  终态路径上的符号串集合

② ○  $A_i$  取 “初态  $\rightarrow$  状态  $i$  路径上的符号串集合” 是否可以？变换规则如何修改？文法变成什么样？

$$\left. \begin{array}{l} \circ \ A_j \rightarrow A_i a \\ \circ \ A_0 \rightarrow \epsilon \end{array} \right\}$$

$$A_0 \rightarrow A_0 a, A_1 \rightarrow A_0 a$$

$$A_0 \rightarrow A_0 b, A_2 \rightarrow A_1 b$$

$$A_3 \rightarrow A_2 b, A_0 \rightarrow \epsilon$$

两种转换方式.



# 为什么还需要正规式？

---

1. 词法规则很简单，正规式描述能力足够
  2. 正规式更简洁、更容易理解
  3. 能更自动构造更高效的词法分析器
  4. 使编译器前端更模块化
- 词法、语法规则的划分没有固定准则
    - 正规式更适合描述标识符、常量、关键字...的结构
    - CFG更适合描述单词的结构化联系、层次化结构，如括号匹配，if-then-else, ...



## 4.3.2 CFG的验证

---

- 证明CFG G生成语言L

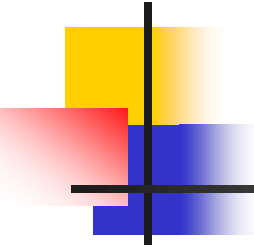
- G生成的每个符号串都在L中

- L中每个符号串都可由G生成

- 例4.7: 验证CFG

$S \rightarrow (S)S \mid \varepsilon$

生成的语言L={ 所有括号组成的, 且括号匹配的字符串, 且只有这些字符串 }



# 一、S推导出的句子都 $\in L$

数学归纳法（推导步数）：

1. 基本情况：一步推导， $\varepsilon$ ，括号匹配
2. 假定步骤 $< n$ 的推导都生成括号匹配的句子，考虑步骤 $= n$ 的最左推导，必形如

$$S \Rightarrow \underbrace{(S)}_x S \xRightarrow{*} \underbrace{(x)}_y S \xRightarrow{*} (x)y$$

$x$ 、 $y$ 为步骤 $< n$ 的推导生成的句子——括号匹配的，因此 $(x)y$ 是括号匹配的

综合1、2，一得证

## 二、L中的符号串S都可推导出

数学归纳法（符号串长度）

1. 基本情况：空串，可由S推导出
2. 假定L中 $<2n$ 的符号串都可由S推导出。考虑长度 $=2n$ 的符号串 $w$ 。  
它必以‘(’开始，设 $(x)$ 为 $w$ 的最短的括号匹配的前缀，则 $w$ 形如 $(x)y$ ， $x$ 、 $y$ 长度小于 $2n$ ，且括号匹配，因此可由S推导出，则存在推导
$$S \Rightarrow (S)S \xRightarrow{*} (x)S \xRightarrow{*} (x)y$$

由1、2，二得证

由一、二，原命题得证

# 设计CFG练习

○ 基本的递归

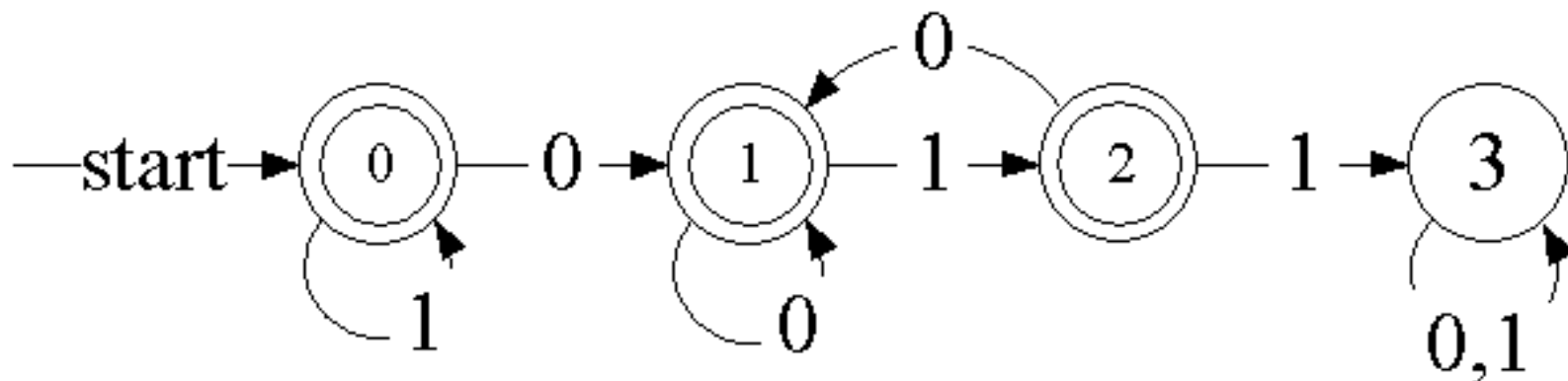
○  $L = \{ a^n b b^{2n} \mid n \geq 0 \}$

$S \Rightarrow b \mid a S b b$

正例可转为CFG

$S \rightarrow b \mid a S b b$

# 不包含子串011的0/1串



$$S \rightarrow 0 A \mid 1 S \mid \varepsilon$$

NFA  $\Rightarrow$  CFG

$$A \rightarrow 0 A \mid 1 B \mid \varepsilon$$

$$B \rightarrow 0 A \mid \varepsilon$$



# 形如 $xy(x \neq y)$ 的01串

- 长度是什么情况必然不是 $xx$ ?  
奇数

- 如何描述?

$S \rightarrow B \mid BSB$

$B \rightarrow 0 \mid 1$

- 其他情况如何描述?

奇数  $\Rightarrow S \rightarrow B \mid BSB$   
 $B \rightarrow 0 \mid 1$

010 110  
x y

偶数

用2个长度为奇数的拼起来



# 设计CFG的难点

---

- 手工进行，无形式化方法
- 不同的语法分析方法对CFG有不同的特殊要求
  - 如自顶向下分析方法和自底向上分析方法
  - CFG设计完成后可能需要修改



# CFG的修改

---

## ○ 两个目的

- 去除“错误”

- 重写，满足语法分析算法要求

不合要求  
的问题

- 二义性

-  $\epsilon$ -moves

- 回路

- 左递归

- 提取左公因子



## 4.3.3 消除二义性

---

○ 例子：条件分支语句

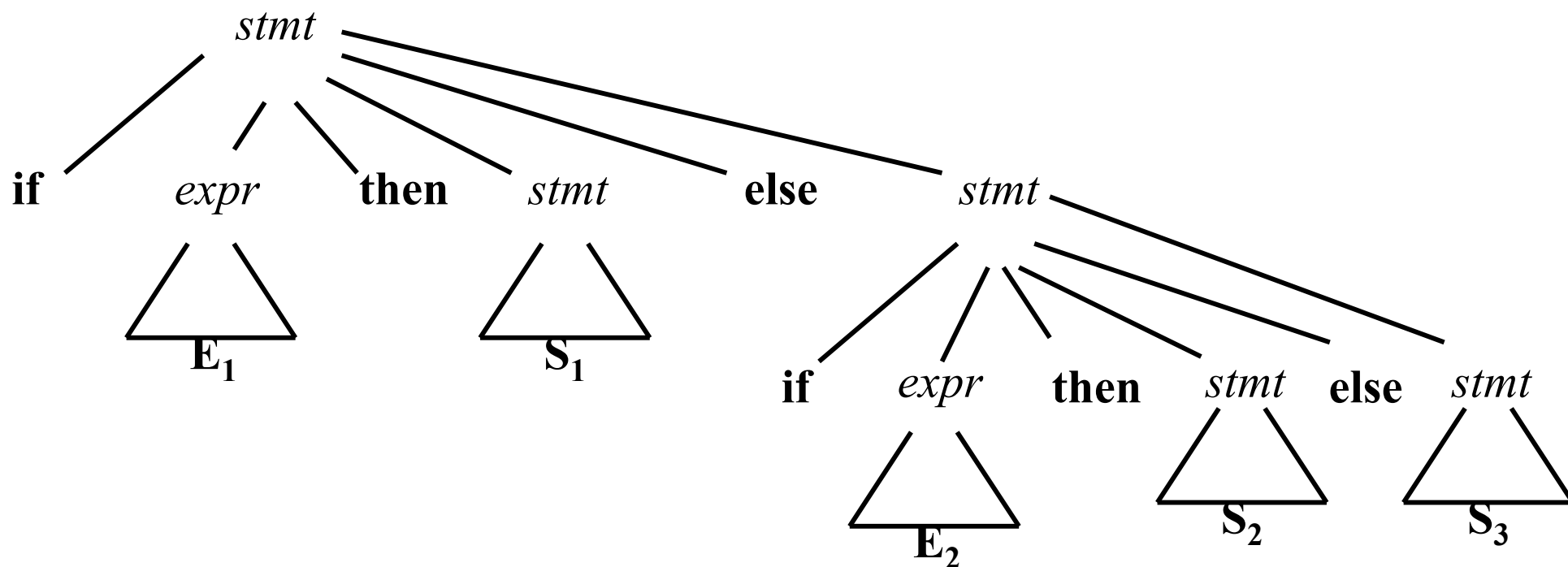
*stmt*  $\rightarrow$  **if** *expr* **then** *stmt*

| **if** *expr* **then** *stmt* **else** *stmt*

| **other** (任何其他形式的语句)

# 无二义性的句子

**if  $E_1$  then  $S_1$  else if  $E_2$  then  $S_2$  else  $S_3$**  语法树如下





# 二义性句子

---

**if  $E_1$  then if  $E_2$  then  $S_1$  else  $S_2$** 有两种意义

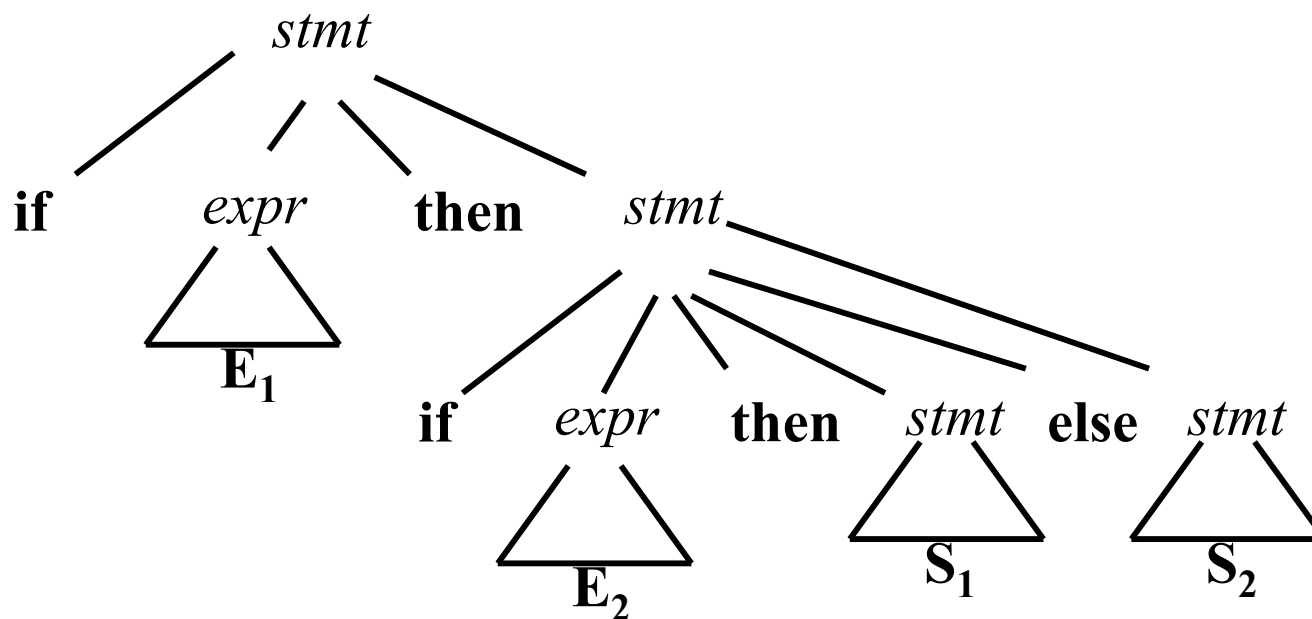
**if  $E_1$  then**  
**{ if  $E_2$  then**  
**$S_1$**   
**{ else**  
**$S_2$**

**vs.**

**{ if  $E_1$  then**  
**if  $E_2$  then**  
**$S_1$**   
**{ else**  
**$S_2$**

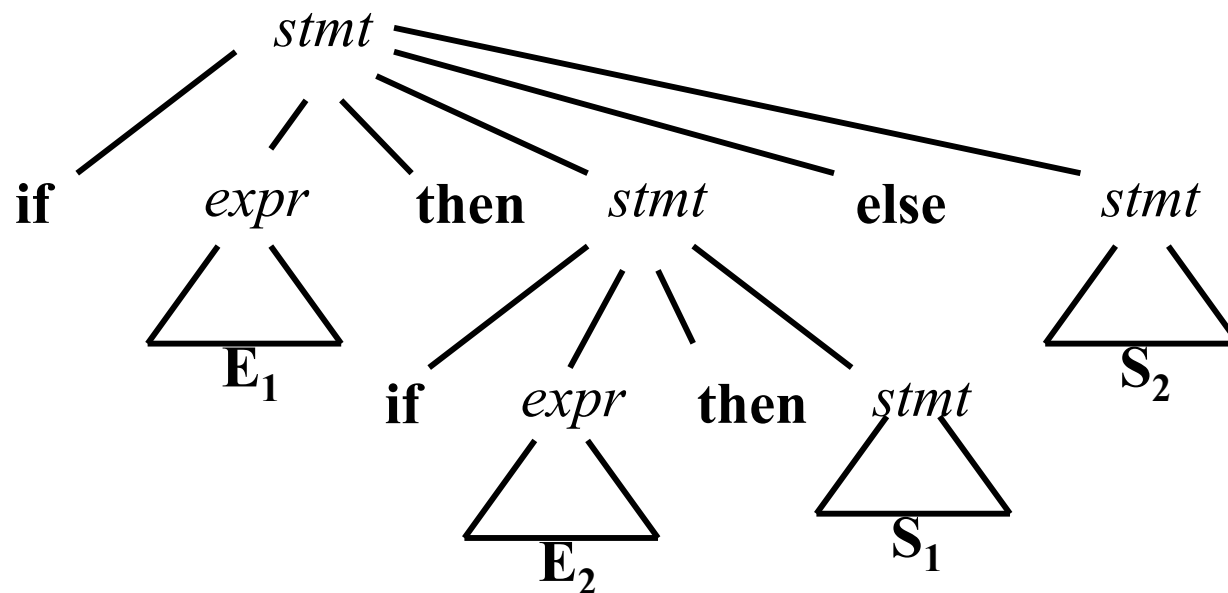
# 两个语法树

**if  $E_1$  then if  $E_2$  then  $S_1$  else  $S_2$**

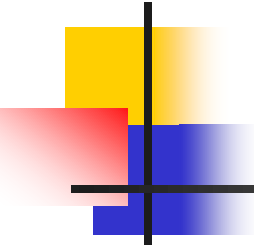


## 两个语法树（续）

**if  $E_1$  then if  $E_2$  then  $S_1$  else  $S_2$**







# 消除二义性

---

- “else与最近的未匹配的then相匹配”
- 修改文法—then和else间的语句必须平衡

$$\begin{aligned} stmt &\rightarrow matched\_stmt \\ &\quad | unmatched\_stmt \end{aligned}$$
$$\begin{aligned} matched\_stmt &\rightarrow \mathbf{if\ expr\ then\ matched\_stmt\ else\ matched\_stmt} \\ &\quad | \mathbf{other} \end{aligned}$$
$$\begin{aligned} unmatched\_stmt &\rightarrow \mathbf{if\ expr\ then\ stmt} \\ &\quad | \mathbf{if\ expr\ then\ matched\_stmt\ else\ unmatched\_stmt} \end{aligned}$$

## 4.3.4 消除左递归

○  $A \xRightarrow{+} A\alpha$

○ 自顶向下分析方法无法处理，死循环

△ ○ 直接左递归的消除

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

任何 $\beta_i$ 均不以A开头，改写为：

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \varepsilon$$

$$A \rightarrow A\alpha \mid \beta$$

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$



## 例4.8

---

$$\mathbf{E} \rightarrow \mathbf{E} + \mathbf{T} \mid \mathbf{T} \quad \longrightarrow \quad \left\{ \begin{array}{l} \mathbf{E} \rightarrow \mathbf{T}\mathbf{E}' \\ \mathbf{E}' \rightarrow + \mathbf{T}\mathbf{E}' \mid \varepsilon \end{array} \right.$$

$$\mathbf{T} \rightarrow \mathbf{T} * \mathbf{F} \mid \mathbf{F} \quad \longrightarrow \quad \left\{ \begin{array}{l} \mathbf{T} \rightarrow \mathbf{F}\mathbf{T}' \\ \mathbf{T}' \rightarrow * \mathbf{F}\mathbf{T}' \mid \varepsilon \end{array} \right.$$

$$\mathbf{F} \rightarrow (\mathbf{E}) \mid \mathbf{id} \quad \longrightarrow \quad \mathbf{F} \rightarrow (\mathbf{E}) \mid \mathbf{id}$$



## 算法4.1：消除间接左递归

输入：CFG  $G$ ，无环路，无 $\epsilon$ 产生式

输出：等价的、无左递归的文法

1. 非终结符按顺序排列 $A_1, A_2, \dots, A_n$
2. for ( $i = 1; i < n; i++$ )  
    for ( $j = 1; j < i - 1; j++$ ) {  
        将所有形如 $A_i \rightarrow A_j \gamma$ 的产生式替换为  
         $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$ ，其中  
         $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$ 为其他对 $A_j$ 的产生式  
    }  
3. 消除所有直接左递归

## 例4.9

$$\left. \begin{array}{l} S \rightarrow Aa \mid b \\ A \rightarrow Ac \mid Sd \mid \varepsilon \end{array} \right\} S \Rightarrow Aa \Rightarrow Sda$$

1. 间接左递归  $\rightarrow$  直接左递归:

$$A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$$

2. 消除直接左递归

$$S \rightarrow Aa \mid b$$

$$A \rightarrow bdA' \mid A'$$

$$A' \rightarrow cA' \mid adA' \mid \varepsilon$$

$$A \rightarrow A\alpha \mid \beta$$

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' \mid \varepsilon$$

## 补充：消除 $\varepsilon$ 产生式

○ 方法：利用产生式进行代入

○  $A \rightarrow \varepsilon, B \rightarrow uAv \Rightarrow B \rightarrow uv \mid uAv$

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

$$F \rightarrow ( E ) \mid \text{id}$$

$$E \rightarrow TE' \mid T$$

$$E' \rightarrow + TE' \mid + T$$

$$T \rightarrow FT' \mid F$$

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

$$F \rightarrow ( E ) \mid \text{id}$$

## 补充：消除 $\varepsilon$ 产生式（续）

$$A_1 \rightarrow A_2 a \mid b$$

$$A_2 \rightarrow bd A_2' \mid A_2'$$

$$A_2' \rightarrow c A_2' \mid bd A_2' \mid \varepsilon$$

$$A_1 \rightarrow A_2 a \mid b \mid a$$

$$A_2 \rightarrow bd A_2' \mid A_2'$$

$$\mid bd$$

$$A_2' \rightarrow c A_2' \mid bd A_2'$$

$$\mid c \mid bd$$

增加即可

# 补充：消除回路

$$S \rightarrow SS \mid (S) \mid \varepsilon$$

$$\text{回路: } S \Rightarrow SS \Rightarrow S$$

$\underbrace{SS}_{S \rightarrow \varepsilon}$

- 如何消除回路？
- 保证每个产生式都加入终结符（开始符号的 $\varepsilon$ 产生式除外）
- 上面文法改写为：  
$$S \rightarrow S(S) \mid (S) \mid \varepsilon$$

$$S \Rightarrow SS \Rightarrow S$$

$$S \Rightarrow S(S) \Rightarrow \varepsilon(S) \Rightarrow (S)$$



## 4.3.5 提取左公因子

把左公因子消除

- 预测分析方法要求——  
向前搜索一个单词，即可确定产生式
- $stmt \rightarrow \text{if } expr \text{ then } stmt \text{ else } stmt$   
|  $\text{if } expr \text{ then } stmt$  不符合!
- 一般的  
 $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$   
改写为  
 $A \rightarrow \alpha A'$   
 $A' \rightarrow \beta_1 \mid \beta_2$



## 算法4.2 提取左公因子

---

输入：CFG  $G$

输出：等价的、提取了左公因子的文法

方法：

对每个非终结符 $A$ ，寻找多个候选式公共的最长前缀 $\alpha$ ，若 $\alpha \neq \varepsilon$ ，则将所有 $A$ 的候选式

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma$ （ $\gamma$ 表示所有其他候选式），改写为

$A \rightarrow \alpha A' \mid \gamma$

$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$



## 例4.10

---

$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

**$i \rightarrow \text{if}$ ,  $t \rightarrow \text{then}$ ,  $e \rightarrow \text{else}$ ,  $E \rightarrow \text{表达式}$ ,  $S \rightarrow \text{语句}$**

改写为:

$$S \rightarrow iEtSS' \mid a$$
$$S' \rightarrow eS \mid \varepsilon$$
$$E \rightarrow b$$

## 4.3.6 CFG无法描述的语言结构

例4.11:  $L_1 = \{ w cw \mid w \in (a \mid b)^* \}$

- 检查标识符必须在使用之前定义
- 语义分析

例4.12:  $L_2 = \{ a^n b^m c^n d^m \mid n \geq 1 \text{ 且 } m \geq 1 \}$

- 检查函数的形参（声明）与实参（调用）的数目是否匹配
- 语法定义一般不考虑参数数目

参数数目.



# CFG无法描述的语言结构(续)

例4.13:  $L_3 = \{ a^n b^n c^n \mid n \geq 0 \}$

- 排版软件，文本加下划线：n个字符，n个退格，n个下划线
- 另一种方式：字符—退格—下划线三元组序列， $(abc)^*$

# 类似语言可用CFG描述

- $L_1' = \{ wcw^R \mid w \in (a \mid b)^*, w^R \text{为} w \text{的反转} \}$   
 $S \rightarrow aSa \mid bSb \mid c$  反转可以.
- $L_2' = \{ a^n b^m c^m d^n \mid n \geq 1 \text{ 且 } m \geq 1 \}$   $a^n b^m c^n d^m$   
 $S \rightarrow aSd \mid aAd$        $A \rightarrow bAc \mid bc$  不行  
 $L_2'' = \{ a^n b^n c^m d^m \mid n \geq 1 \text{ 且 } m \geq 1 \}$   
 $S \rightarrow AB$        $A \rightarrow aAb \mid ab$        $B \rightarrow cBd \mid cd$
- $L_3' = \{ \underline{a^n b^n} \mid n \geq 0 \}$   
 $S \rightarrow aSb \mid ab$  2个可以       $a^n b^n c^n$  不行



# $L_3'$ 用正规式无法描述

---

假定存在DFA  $D$  接受  $L_3'$ , 其状态数为  $k$

设状态  $s_0, s_1, \dots, s_k$  为读入  $\varepsilon, a, aa, \dots, a^k$  后的状态

$\rightarrow s_i$  为读入  $i$  个  $a$  达到的状态 ( $0 \leq i \leq k$ )

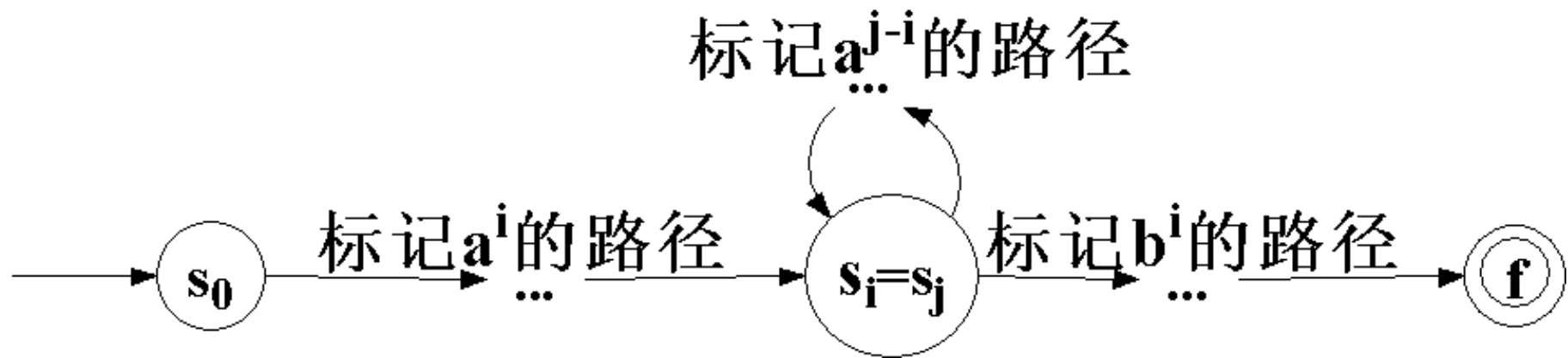
总状态数  $k \rightarrow s_0, s_1, \dots, s_k$  中至少有两个相同状态,  
不妨设为  $s_i, s_j, i < j$

(不做要求)

## $L_3'$ 用正规式无法描述 (续)

$a^i b^i \in L_3' \rightarrow s_i (s_j)$  到终态路径标记为  $b^i$

$\rightarrow$  初态  $\rightarrow$  终态还有标为  $a^i b^i$  的路径  $\rightarrow D$  接受  $a^i b^i$ ,  
矛盾!







## 4.4 自顶向下语法分析

---

○ 确定输入串的一个最左推导

□ 总是替换最左NT

→ 语法树的构造由左至右

→ 与输入串的扫描顺序一致

□  $A \Rightarrow aBc \Rightarrow adDc \Rightarrow adec$  (扫描a, 扫描d, 扫描e, 扫描c – 接受!)



# 学习内容

---

- 递归下降分析, recursive-descent parsing
- 预测分析, predictive parsing, 无回溯
- 错误恢复
- 实现方法



## 4.4.1 递归下降分析方法

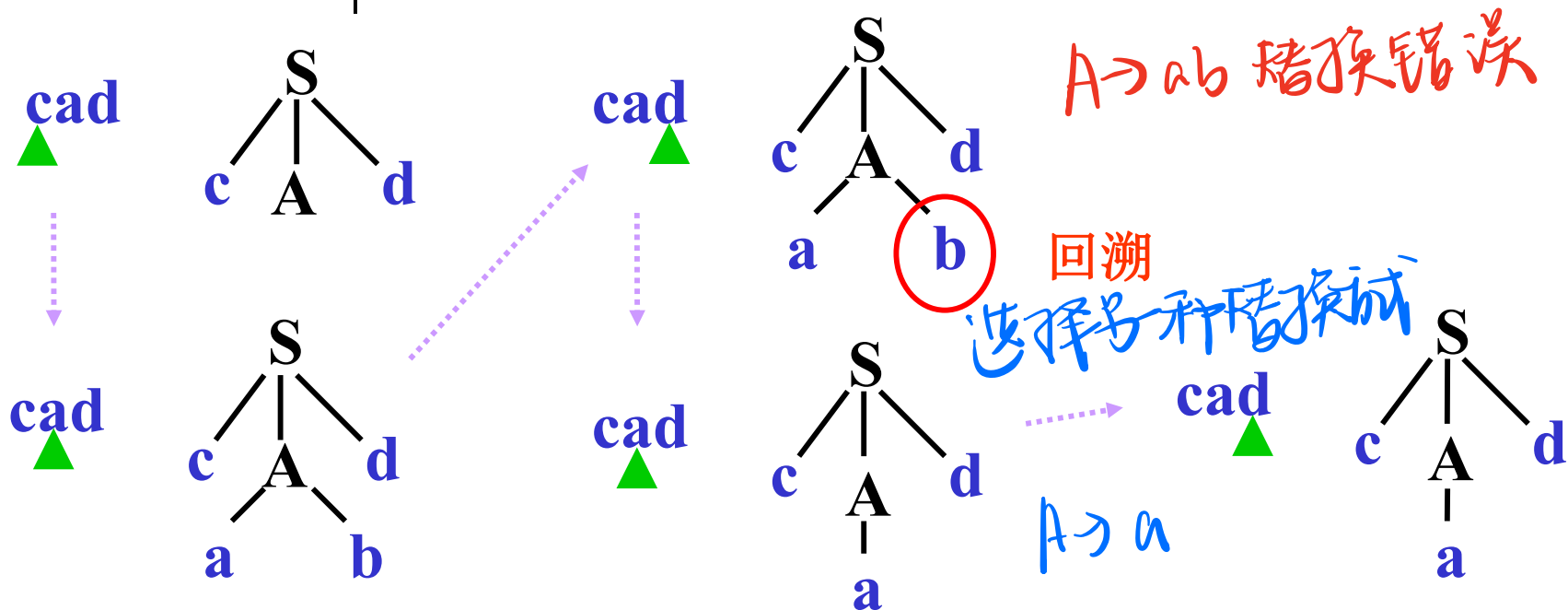
---

- 自顶向下分析方法的一般策略
- 根据输入符号选择产生式
- 选择错误，需要回溯
- 分析程序语言结构，回溯很少发生

## 例4.14

$$S \rightarrow c A d$$
$$A \rightarrow ab \mid a$$

输入: cad





## 4.4.2 预测分析方法

- 无需回溯的递归下降法，需改写文法

  - 消除左递归

  - △ □ 提取左公因子 *前面提取左因子算法*

- “当前输入符号” + “待扩展非终结符” →

唯一确定应用哪个产生式



## 4.4.3 利用状态转换图

### ○ 特点

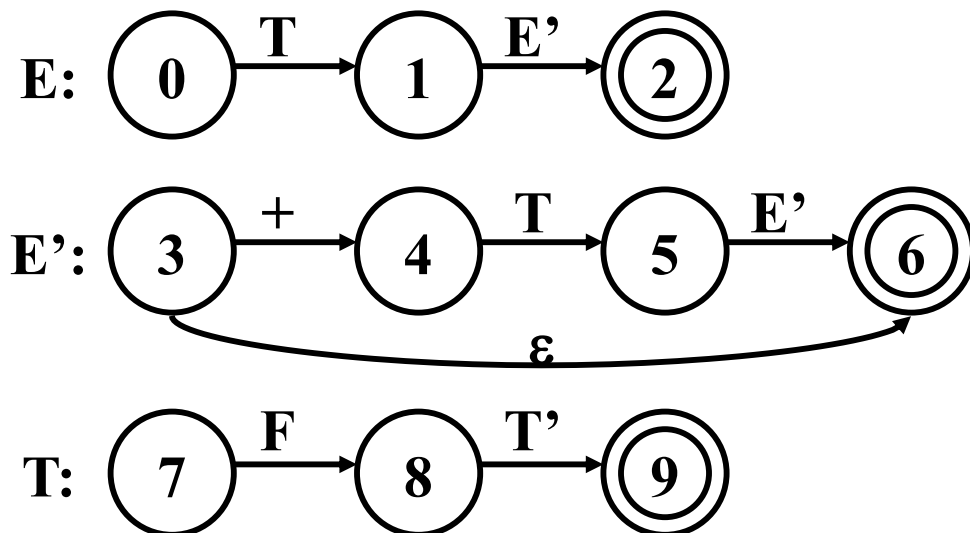
- 非终结符 $\leftrightarrow$ 状态转换图
- 边的标记
  - 单词：与下一个输入符号匹配 $\rightarrow$ 状态转换
  - 非终结符：对其进行扩展（调用对应函数）

### ○ 创建NT A对应TD:

1. 创建一个初态和一个终态
2. 对每个产生式 $A \rightarrow X_1X_2...X_n$ ，创建初态到终态的一条路径，边标记为 $X_1, X_2, \dots, X_n$

## 例4.15

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



如何使用TD进行语法分析？

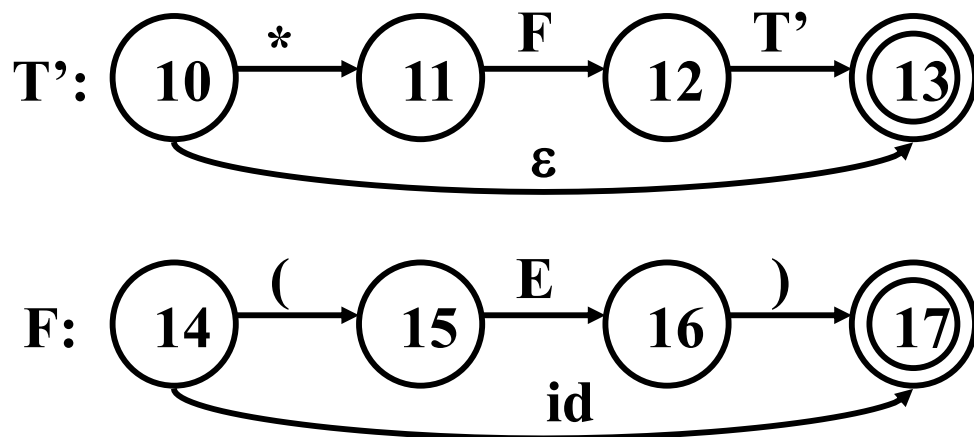
$\varepsilon$ 边如何处理？

TD是否可以化简？

化简的重要性？

## 例4.15

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



如何使用TD进行语法分析？

ε边如何处理？

TD是否可以化简？

化简的重要性？





# 使用TD进行语法分析

算法:

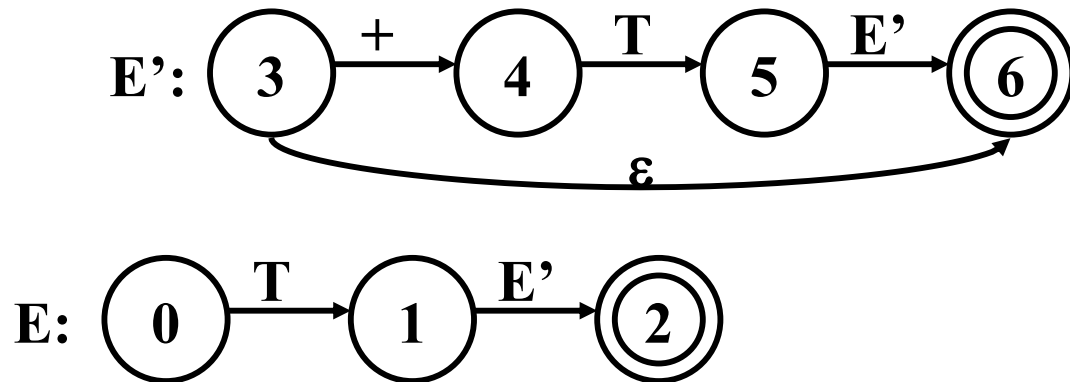
1. TD当前状态为s, 下个输入符号为a, 存在边  $s \xrightarrow{a} t$   $\rightarrow$  输入指针前移, 当前状态  $\rightarrow t$
2. 存在边  $s \xrightarrow{NT A} t$   $\rightarrow$  当前状态  $\rightarrow A$  的TD的初态, 输入指针不变。当到达A的终态, 立即转换到t——读入“A”使得从s转换到t
3. 存在边  $s \xrightarrow{\epsilon} t$   $\rightarrow$  当前状态  $\rightarrow t$ , 输入指针不变。

# 使用TD（代码示例）

```
main()  
{  
    TD_E();  
}
```

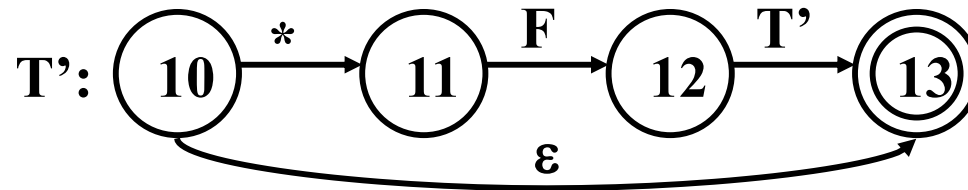
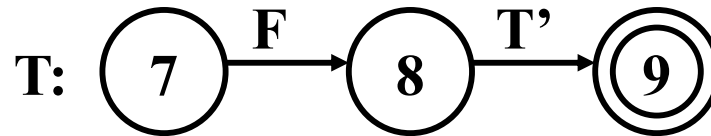
```
TD_E()  
{  
    TD_T();  
    TD_E'();  
}
```

```
TD_E'()  
{ token = get_token();  
  if token = '+' then  
    { TD_T(); TD_E'(); } }
```



# 使用TD（代码示例）

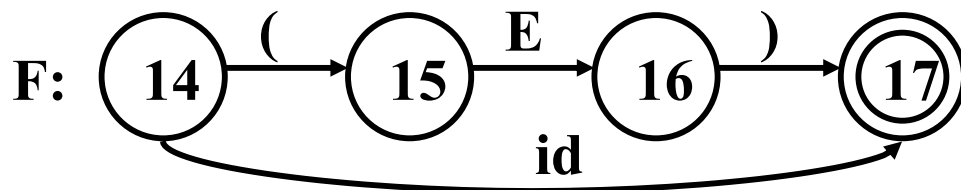
```
TD_T()  
{  
    TD_F();  
    TD_T'();  
}
```



```
TD_T'()  
{ token = get_token();  
  if token = '*' then  
    { TD_F(); TD_T'(); } }
```

# 使用TD（代码示例）

```
TD_F()  
{ token = get_token();  
  if token = '(' then  
    { TD_E(); match('('); }  
  else  
    if token.value <> id then  
      {error + EXIT}  
    else  
      ... }  
}
```

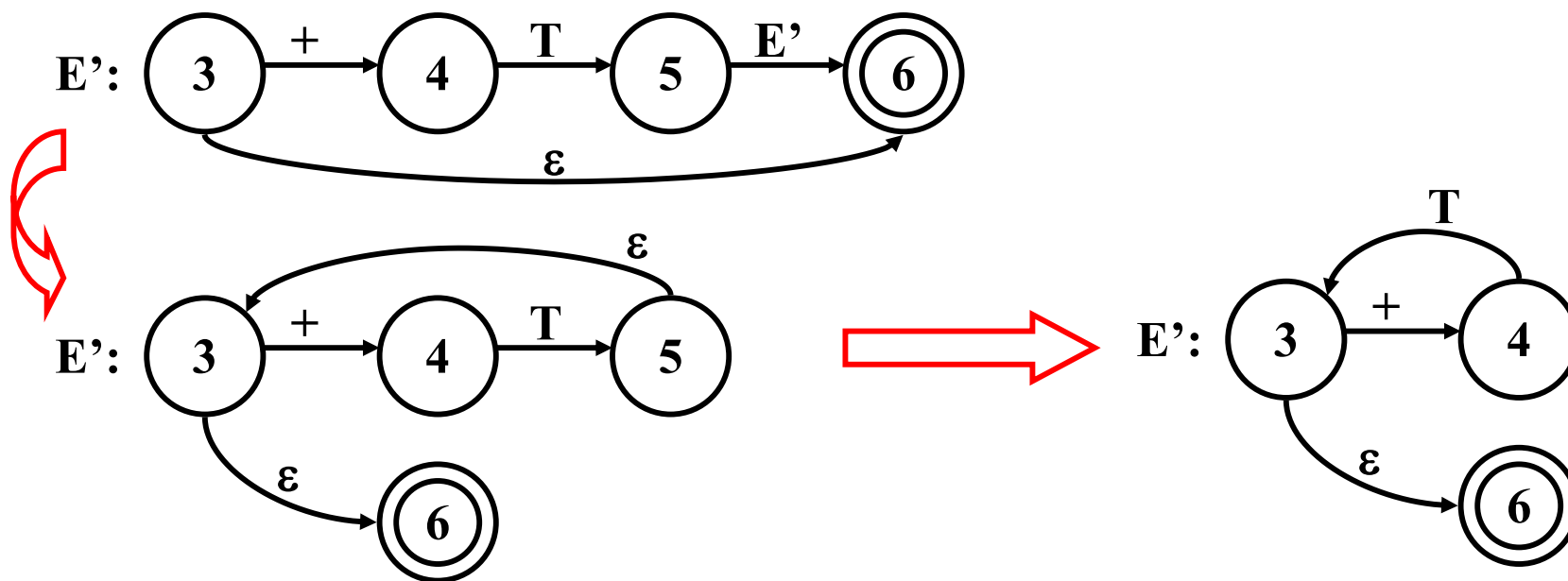


**$\epsilon$ 边如何处理?**

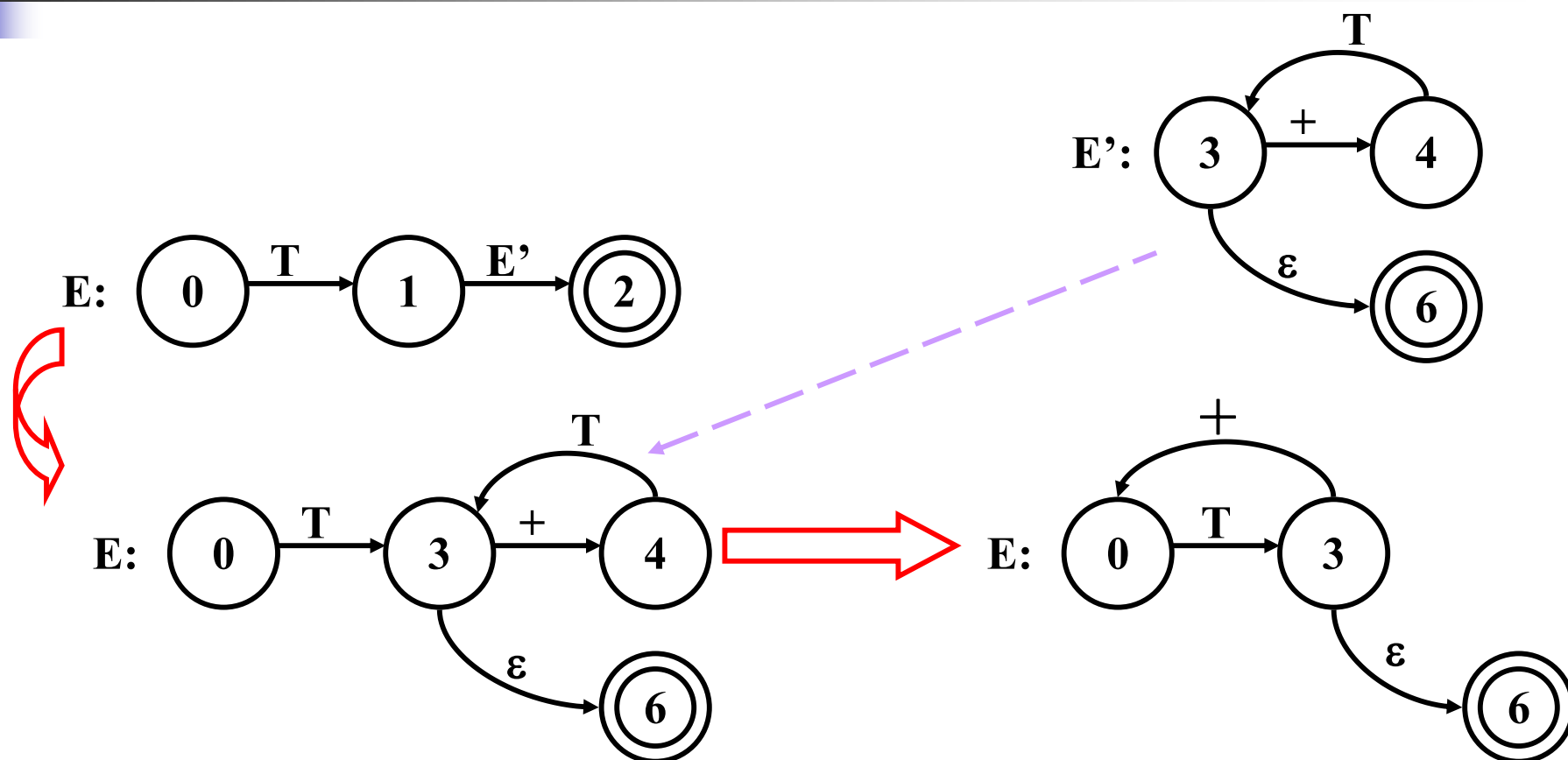
**... “else unget()  
and terminate”**

**NOTE:** 并未给出所有错误的处理代码

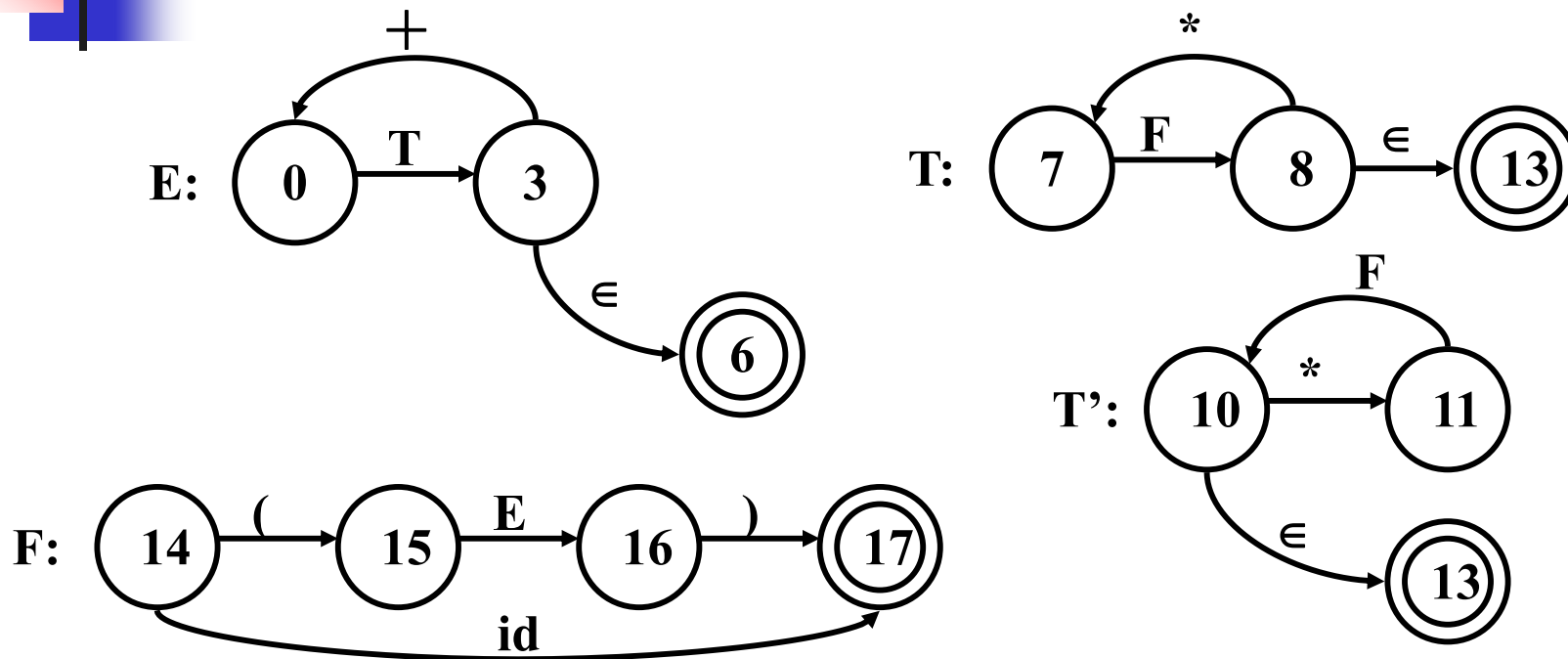
# TD的化简



# TD的化简

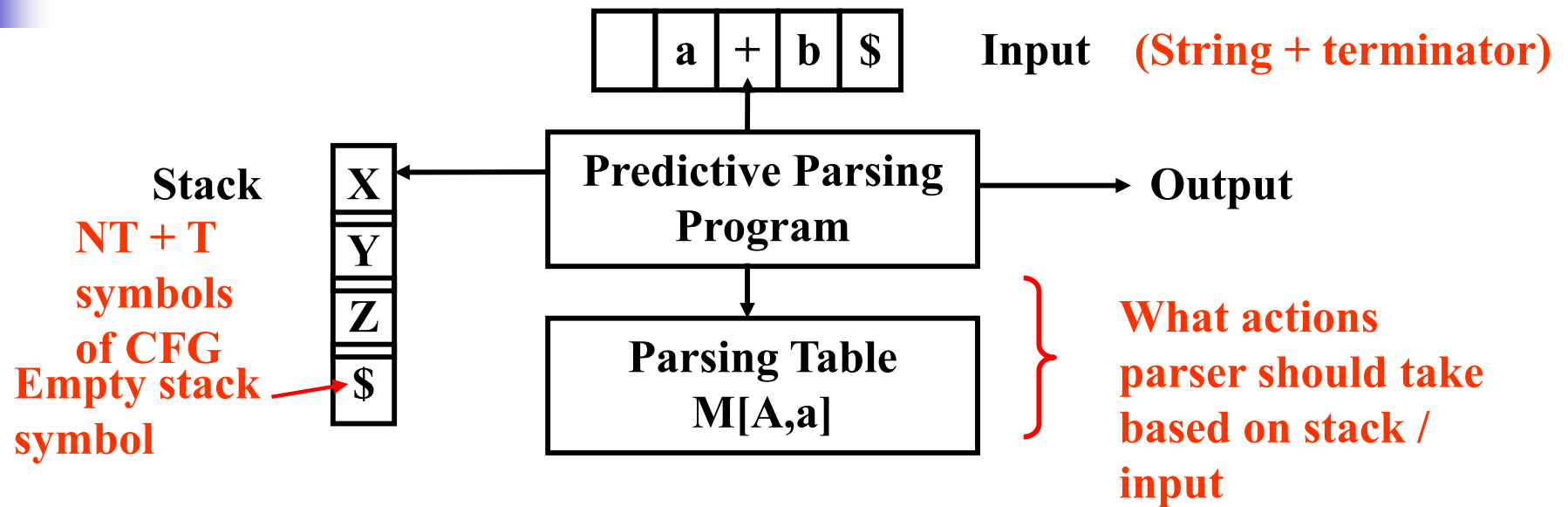


# 完整的化简



化简的重要性？  
——代码的变化？速  
度提高20%-25%

## 4.4.4 非递归预测分析方法



- 输入缓冲、栈、预测分析表、输出流
  - 栈：语法符号序列，栈底符 $\$$
  - 预测分析表：二维数组 $M[A, a]$ ， $A$ 为NT， $a$ 为T或 $\$$ ，其值为某种动作

初始的栈

初始符号
$\$$





# 预测分析器运行方法

---

- 考虑栈顶符号 $X$ ，当前输入符号 $a$ 
  1.  $X=a=\$$ ，终止，接受输入串
  2.  $X=a\neq \$$ ， $X$ 弹出栈，输入指针前移
  3.  $X$ 为NT:
    - $M[X, a] = \{X \rightarrow UVW\}$ ，将栈中 $X$ 替换为 $UVW$ （ $U$ 在栈顶），输出可以是打印出产生式，表示推导
    - $M[X, a] = \text{error}$ ，调用错误恢复函数



## 算法4.3 非递归预测分析方法

输入：符号串 $w$ ，文法 $G$ 及其预测分析表 $M$

输出：若 $w \in L(G)$ ， $w$ 的一个最左推导；否则，错误提示  
方法：

初始：栈中为 $SS$ （ $S$ 在栈顶），输入缓冲区为 $w\$$ 。分  
析器运行算法：

设置 $ip$ 指向输入缓冲区的第一个符号；

do {

    令 $X$ 为栈顶符号， $a$ 为 $ip$ 指向符号

    if ( $X$ 为终结符或 $\$$ ) {



## 算法4.3（续）

---

```
    if (X == a) {  
        X弹出栈, ip前移;  
    }  
    else error();  
} else if (M[X, a] =  $X \rightarrow Y_1 Y_2 \dots Y_k$ ) {  
    X弹出栈;  
    将 $Y_k, Y_{k-1}, \dots, Y_1$ 压栈,  $Y_1$ 置于栈顶;  
    输出产生式 $X \rightarrow Y_1 Y_2 \dots Y_k$ ;  
} else error();  
} while (X != $);
```

# 例4.16

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

**Table M**

Non-terminal	INPUT SYMBOL					
	id	+	*	(	)	\$
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	$F \rightarrow id$			$F \rightarrow (E)$		

# 例4.16 (续)

STACK	INPUT	OUTPUT
\$E	id + id * id\$	
\$E'T	id + id * id\$	$E \rightarrow TE'$
\$E'T'F	id + id * id\$	$T \rightarrow FT'$
\$E'T'id	id + id * id\$	$F \rightarrow id$
\$E'T'	+ id * id\$	
\$E'	+ id * id\$	$T' \rightarrow \epsilon$
\$E'T+	+ id * id\$	$E' \rightarrow \underline{+}TE'$
\$E'T	id * id\$	
\$E'T'F	id * id\$	$T \rightarrow FT'$
\$E'T'id	id * id\$	$F \rightarrow id$
\$E'T'	* id\$	
\$E'T'F*	* id\$	$T' \rightarrow \underline{*}FT'$
\$E'T'F	id\$	
\$E'T'id	id\$	$F \rightarrow id$
\$E'T'	\$	
\$E'	\$	$T' \rightarrow \epsilon$
\$	\$	$E' \rightarrow \epsilon$

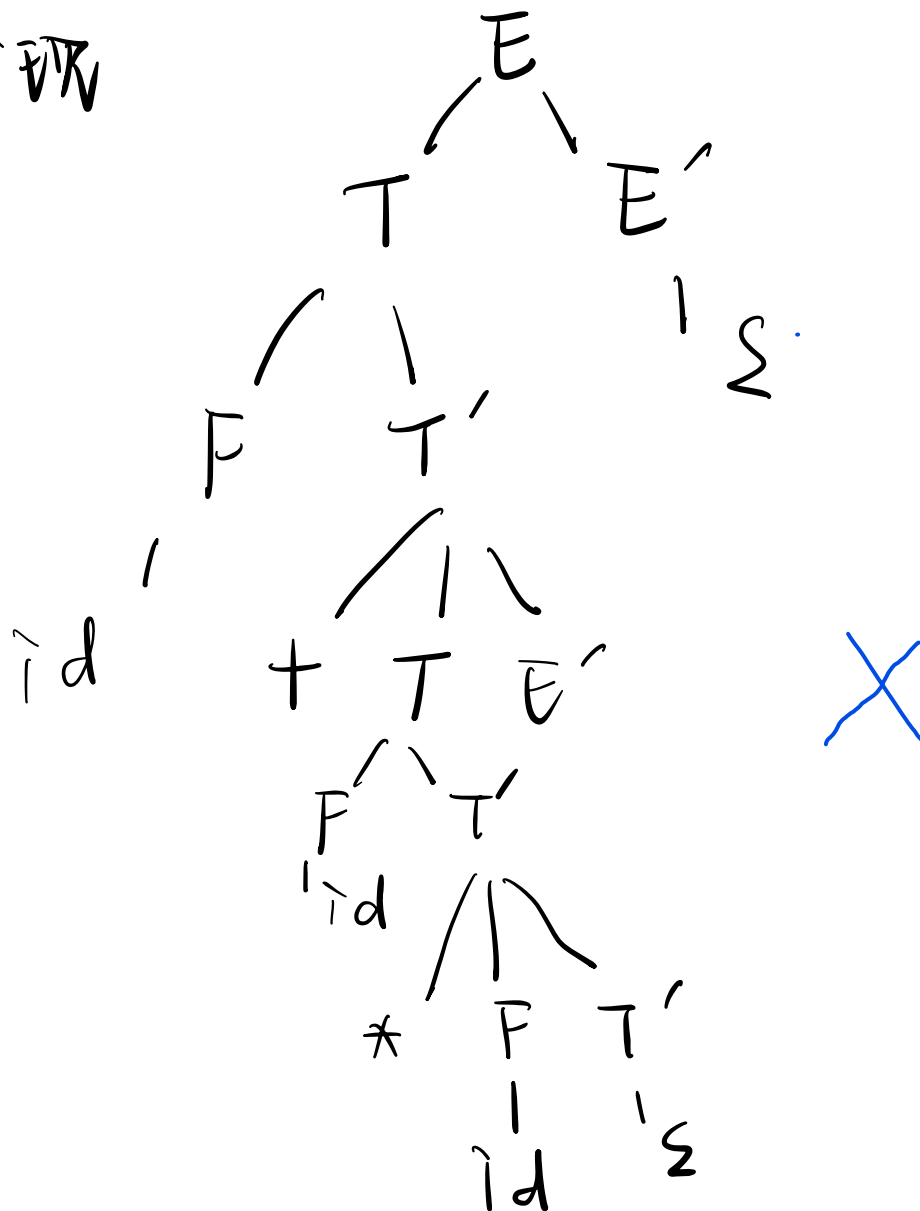
*me E id]*

**Expend Input**

栈	输入	输出
$\$E$	$id + id * id\$$	
$\$E'T$	$id + id * id\$$	$E \rightarrow TE'$
$\$E'T'F$	$id + id * id\$$	$T \rightarrow FT'$
$\$E'T'id$	$id + id * id\$$	$F \rightarrow id$
$\$E'T'$	$+ id * id\$$	
$\$E'$	$+ id * id\$$	$T' \rightarrow \epsilon$
$\$E'T +$	$+ id * id\$$	$E' \rightarrow +TE'$
$\$E'T$	$id * id\$$	
$\$E'T'F$	$id * id\$$	$T \rightarrow FT'$
$\$E'T'id$	$id * id\$$	$F \rightarrow id$
$\$E'T'$	$* id\$$	
$\$E'T'F*$	$* id\$$	$T' \rightarrow *FT'$
$\$E'T'F$	$id\$$	
$\$E'T'id$	$id\$$	$F \rightarrow id$
$\$E'T'$	$\$$	
$\$E'$	$\$$	$T' \rightarrow \epsilon$
$\$$	$\$$	$E' \rightarrow \epsilon$

图4-16 预测语法分析器在输入 $id+id*id$ 上所做的移动

可以发现





## 4.4.5 FIRST和FOLLOW

---

- 如何构造预测分析表？

- 计算FIRST和FOLLOW函数
- 应用构造算法

- FIRST？

- $\text{FIRST}(\alpha)$ :  $\alpha \in (T \cup NT)^*$

- 所有 $\alpha$ 可推导出的符号串的开头终结符的集合

- $\alpha \xRightarrow{*} \epsilon \rightarrow \epsilon \in \text{FIRST}(\alpha)$





# FIRST和FOLLOW

---

## ○ FOLLOW?

□ FOLLOW(A):  $A \in NT$

➤ 所有句型中紧接A之后的终结符的集合

➤  $S \xRightarrow{*} \alpha A a \beta \rightarrow a \in \text{FOLLOW}(A)$

➤  $S \xRightarrow{*} \alpha A \rightarrow \$ \in \text{FOLLOW}(A)$

如何计算FIRST

# 计算单个符号的FIRST函数

1. 若 $X$ 是终结符, 则 $\text{FIRST}(X)=\{X\}$

2. 若 $X \rightarrow \varepsilon$ , 则将 $\varepsilon$ 加入 $\text{FIRST}(X)$

3. 若 $X \rightarrow Y_1 Y_2 \dots Y_k$ , 则

$\text{FIRST}(Y_1)$ 加入 $\text{FIRST}(X)$

若 $Y_1 \xRightarrow{*} \varepsilon$

$\text{FIRST}(Y_2)$ 加入 $\text{FIRST}(X)$

不可为空 计算 $Y_2$

若 $Y_2 \xRightarrow{*} \varepsilon$

$\text{FIRST}(Y_3)$ 加入 $\text{FIRST}(X)$

...

若 $Y_{k-1} \xRightarrow{*} \varepsilon$   $\text{FIRST}(Y_k)$ 加入 $\text{FIRST}(X)$

若 $Y_k \xRightarrow{*} \varepsilon$

$\varepsilon$ 加入 $\text{FIRST}(X)$

NOTE: 一旦 $Y_i \not\xRightarrow{*} \varepsilon$ , 即停止

○ 重复1—3, 直至所有符号的FIRST集都不再变化



# 计算符号串的FIRST函数

$\text{FIRST}(X_1 X_2 \dots X_n) = \text{FIRST}(X_1) \text{ “+”}$

$\text{FIRST}(X_2)$  if  $\varepsilon$  is in  $\text{FIRST}(X_1)$  “+”

$\text{FIRST}(X_3)$  if  $\varepsilon$  is in  $\text{FIRST}(X_2)$  “+”

...

$\text{FIRST}(X_n)$  if  $\varepsilon$  is in  $\text{FIRST}(X_{n-1})$

注意：仅当对所有 $i$ ,  $\varepsilon \in \text{FIRST}(X_i)$ , 才将 $\varepsilon$ 加入  
 $\text{FIRST}(X_1 X_2 \dots X_n)$

## 例4.17

Computing First for:

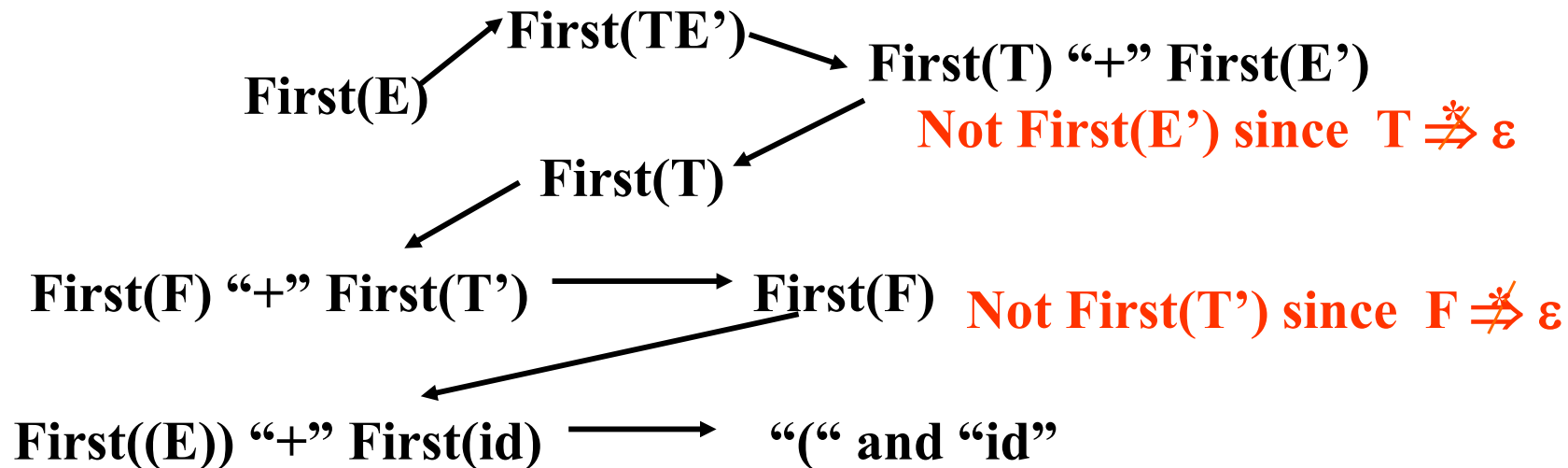
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow ( E ) \mid id$





## 例4.17 最终结果

---

**Overall:  $\text{First}(E) = \{ (, \text{id} \} = \text{First}(F)$**

**$\text{First}(E') = \{ +, \varepsilon \}$      $\text{First}(T') = \{ *, \varepsilon \}$**

**$\text{First}(T) = \text{First}(F) = \{ (, \text{id} \}$**



例

---

**Given the production rules:**

$$S \rightarrow i E t S S' \mid a$$

$$S' \rightarrow eS \mid \varepsilon$$

$$E \rightarrow b$$

**Verify that**

$$\text{First}(S) = \{ i, a \}$$

$$\text{First}(S') = \{ e, \varepsilon \}$$

$$\text{First}(E) = \{ b \}$$

如何计算 FOLLOW

# 计算 FOLLOW(A)——非终结符

$S \xrightarrow{*} S\$$

1.  $\$$  加入 FOLLOW(S), S 为开始符号,  $\$$  为输入串结束标记  
因为  $start - 开始$  为开始符号,  $\$$
2. 若  $A \rightarrow \alpha B \beta$ , 则 FIRST( $\beta$ ) 中符号除  $\epsilon$  外, 均加入 FOLLOW(B)  
因为: 若有  $\beta \xrightarrow{*} a\gamma$ , 显然会有  $S \xrightarrow{*} \delta \alpha B a \gamma \eta$
3. 若  $A \rightarrow \alpha B$  或  $A \rightarrow \alpha B \beta$  且  $\beta \xrightarrow{*} \epsilon$ , FOLLOW(A) 中所有符号加入 FOLLOW(B)  
因为: 若有  $S \xrightarrow{*} \gamma A a \eta$ , 则有  $S \xrightarrow{*} \gamma \alpha B a \eta$

$\alpha, \beta$  为语法符号串

A, B 为非终结符。

# 二次扫描算法计算FOLLOW

1. 对所有非终结符 $X$ , FOLLOW( $X$ )置为空集。

$\triangle$  FOLLOW( $S$ )= $\{\$ \}$ ,  $S$ 为开始符号

2. 重复下面步骤, 直至所有FOLLOW集都不再变化

for 所有产生式 $X \rightarrow X_1 X_2 \dots X_m$

for  $j = 1$  to  $m$

if  $X_j$ 为非终结符, 则

Follow( $X_j$ )=Follow( $X_j$ ) $\cup$ (First( $X_{j+1}, \dots, X_m$ )- $\{\epsilon\}$ );

若 $\epsilon \in \text{First}(X_{j+1}, \dots, X_m)$ 或 $X_{j+1}, \dots, X_m = \epsilon$ , 则

Follow( $X_j$ )=Follow( $X_j$ ) $\cup$ Follow( $X$ );



## 例4.17

Compute Follow for:

$$E \rightarrow \textcircled{T} E'$$

$$E' \rightarrow + T E' \mid \varepsilon$$

$$T \rightarrow F T'$$

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

$$F \rightarrow ( E ) \mid \text{id}$$

	FIRST		FOLLOW
E	( id	E	\$ )
E'	$\varepsilon$ +	E'	\$ )
T	( id	$\textcircled{T}$	+ \$ )
T'	$\varepsilon$ *	T'	+ \$ )
F	( id	F	+ * \$ )

# 例

Recall:

$$S \rightarrow i E t SS' \mid a$$

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

$$S' \rightarrow eS \mid \varepsilon$$

$$\text{FIRST}(S') = \{ e, \varepsilon \}$$

$$E \rightarrow b$$

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

**FOLLOW(S)** – 包含\$, S是开始符号

$S \rightarrow i E t SS'$ , 将**FIRST(S')**加入 – 除去 $\varepsilon$

$S' \xRightarrow{*} \varepsilon$ , 将**FOLLOW(S)**加入

$S' \rightarrow eS$ , 将**FOLLOW(S')**加入

因此, **FOLLOW(S) = { e, \$ }**

**Follow(S') = Follow(S) HOW?**

**Follow(E) = { t }**

# FIRST与预测分析

Consider the following derivation:  
**What's First for each non-terminal ?**

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

**id, (  $\in$  First(T), First(F)**

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow \underline{(E)} T'E' \Rightarrow$

$(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\underline{id} T'E') T'E' \Rightarrow$

$(id E') T'E' \Rightarrow (id) T'E' \Rightarrow (id) * FT'E' \Rightarrow$

$(id) * id T'E' \Rightarrow (id) * id E' \Rightarrow (id) * id + TE' \Rightarrow$

$(id) * id + FT'E' \Rightarrow (id) * id + id T'E' \xRightarrow{*} (id) * id + id\$$

# FIRST与预测分析

Consider the following derivation:  
**What's First for each non-terminal ?**

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

**id, (  $\in$  First(F)**

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow \underline{(E)} T'E' \Rightarrow$

$(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\underline{id} T'E') T'E' \Rightarrow$

$(id E') T'E' \Rightarrow (id) T'E' \Rightarrow (id) * FT'E' \Rightarrow$

$(id) * id T'E' \Rightarrow (id) * id E' \Rightarrow (id) * id + TE' \Rightarrow$

$(id) * id + FT'E' \Rightarrow (id) * id + \underline{id} T'E' \xRightarrow{*} (id) * id + id\$$

# FIRST与预测分析

Consider the following derivation:  
What's First for each non-terminal ?

$$\text{First}(T') = \{ *, \epsilon \}$$

$$\begin{aligned} E &\Rightarrow TE' \Rightarrow FT'E' \Rightarrow (E) T'E' \Rightarrow \\ & (TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\text{id } T'E') T'E' \Rightarrow \\ & (\text{id } E') T'E' \Rightarrow (\text{id}) T'E' \Rightarrow (\text{id}) * FT'E' \Rightarrow \\ & (\text{id}) * \text{id } T'E' \Rightarrow (\text{id}) * \text{id } E' \Rightarrow (\text{id}) * \text{id} + TE' \Rightarrow \\ & (\text{id}) * \text{id} + FT'E' \Rightarrow (\text{id}) * \text{id} + \text{id } T'E' \xrightarrow{*} (\text{id}) * \text{id} + \text{id} \$ \end{aligned}$$

Red arrows indicate the application of the rule  $T' \rightarrow \epsilon$  at the following steps:

- From  $(\text{id } T'E') T'E'$  to  $(\text{id}) T'E'$
- From  $(\text{id}) T'E'$  to  $(\text{id}) * FT'E'$
- From  $(\text{id}) * \text{id } T'E'$  to  $(\text{id}) * \text{id } E'$
- From  $(\text{id}) * \text{id} + \text{id } T'E'$  to  $(\text{id}) * \text{id} + \text{id} \$$

# FIRST与预测分析

Consider the following derivation:  
What's First for each non-terminal ?

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

$$\begin{aligned} E &\Rightarrow TE' \Rightarrow FT'E' \Rightarrow (E) T'E' \Rightarrow \\ & (TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\text{id } T'E') T'E' \Rightarrow \\ & (\text{id } E') T'E' \Rightarrow (\text{id}) T'E' \Rightarrow (\text{id}) * FT'E' \Rightarrow \\ & (\text{id}) * \text{id } T'E' \Rightarrow (\text{id}) * \text{id } E' \Rightarrow (\text{id}) * \text{id } \underline{\underline{+}} TE' \Rightarrow \\ & (\text{id}) * \text{id } + FT'E' \Rightarrow (\text{id}) * \text{id } + \text{id } T'E' \xrightarrow{*} (\text{id}) * \text{id } + \text{id } \$ \end{aligned}$$

Red arrows indicate the application of the rule  $E' \rightarrow \varepsilon$  at the following steps:

- From  $(\text{id } E') T'E'$  to  $(\text{id}) T'E'$
- From  $(\text{id}) * \text{id } E'$  to  $(\text{id}) * \text{id } \underline{\underline{+}} TE'$
- From  $(\text{id}) * \text{id } + \text{id } T'E'$  to  $(\text{id}) * \text{id } + \text{id } \$$

# FIRST与预测分析

Consider the following derivation:  
What's First for each non-terminal ?

**First(F) = { (, id }**

$$\begin{aligned} E &\Rightarrow TE' \Rightarrow FT'E' \Rightarrow \underline{(E)} T'E' \Rightarrow \\ & (TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\underline{id} T'E') T'E' \Rightarrow \\ & (id E') T'E' \Rightarrow (id) T'E' \Rightarrow (id) * FT'E' \Rightarrow \\ & (id) * \underline{id} T'E' \Rightarrow (id) * id E' \Rightarrow (id) * id + TE' \Rightarrow \\ & (id) * id + FT'E' \Rightarrow (id) * id + \underline{id} T'E' \xRightarrow{*} (id) * id + id\$ \end{aligned}$$

# FIRST与预测分析

Consider the following derivation:

**What's First for each non-terminal ?**

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow (\underline{E}) T'E' \Rightarrow$

$(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\underline{id} T'E') T'E' \Rightarrow$

$(id E') T'E' \Rightarrow (id) T'E' \Rightarrow (id) \underline{*} FT'E' \Rightarrow$

$(id) \underline{*} \underline{id} T'E' \Rightarrow (id) \underline{*} id E' \Rightarrow (id) \underline{*} id \underline{+} TE' \Rightarrow$

$(id) \underline{*} id + FT'E' \Rightarrow (id) \underline{*} id + \underline{id} T'E' \xRightarrow{*} (id) \underline{*} id + id \underline{\$}$



# FOLLOW与预测分析

Consider the following derivation:

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

What's Follow for each non-terminal ?

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow (E) T'E' \Rightarrow$

$(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (id T'E') T'E' \Rightarrow$

$(id E') T'E' \Rightarrow (id) T'E' \Rightarrow (id) * FT'E' \Rightarrow$

$(id) * \underline{id} T'E' \Rightarrow (id) * id E' \Rightarrow (id) * id + TE' \Rightarrow$

$(id) * id + FT'E' \Rightarrow (id) * id + id T'E' \xRightarrow{*} (id) * id + id\$$

# FOLLOW与预测分析

Consider the following derivation:

**Follow(T) = { + , ) , \$ }**

What's Follow for each non-terminal ?

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow (E) T'E' \Rightarrow$

$(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\text{id } T'E') T'E' \Rightarrow$

$(\text{id } E') T'E' \Rightarrow (\text{id}) T'E' \Rightarrow (\text{id}) * FT'E' \Rightarrow$

$(\text{id}) * \underline{\text{id}} T'E' \Rightarrow (\text{id}) * \text{id } E' \Rightarrow (\text{id}) * \text{id} + TE' \Rightarrow$

$(\text{id}) * \text{id} + FT'E' \Rightarrow (\text{id}) * \text{id} + \text{id } T'E' \xrightarrow{*} (\text{id}) * \text{id} + \text{id} \$$

# FOLLOW与预测分析

Consider the following derivation:

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

What's Follow for each non-terminal ?

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow (E) T'E' \Rightarrow$

$(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (id T'E') T'E' \Rightarrow$

$(id E') T'E' \Rightarrow (id) T'E' \Rightarrow (id) * FT'E' \Rightarrow$

$(id) * \underline{id} T'E' \Rightarrow (id) * id E' \Rightarrow (id) * id + TE' \Rightarrow$

$(id) * id + FT'E' \Rightarrow (id) * id + id T'E' \xRightarrow{*} (id) * id + id\$$

# FOLLOW与预测分析

Consider the following derivation:

What's Follow for each non-terminal ?

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

$\text{Follow}(E') = \{ \$, ) \}$

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow (E) T'E' \Rightarrow$   
 $(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\text{id } T'E') T'E' \Rightarrow$

$(\text{id } E') T'E' \Rightarrow (\text{id}) T'E' \Rightarrow (\text{id}) * FT'E' \Rightarrow$

$(\text{id}) * \text{id } T'E' \Rightarrow (\text{id}) * \text{id } E' \Rightarrow (\text{id}) * \text{id} + TE' \Rightarrow$

$(\text{id}) * \text{id} + FT'E' \Rightarrow (\text{id}) * \text{id} + \text{id } T'E' \xRightarrow{*} (\text{id}) * \text{id} + \text{id} \$$

# FOLLOW与预测分析

Consider the following derivation:

**What's First for each non-terminal ?**

**What's Follow for each non-terminal ?**

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow \underline{(E)} T'E' \Rightarrow$   
 $(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (\underline{id} T'E') T'E' \Rightarrow$   
 $(\underline{id} E') T'E' \Rightarrow (\underline{id}) T'E' \Rightarrow (\underline{id}) * \underline{FT'E'} \Rightarrow$   
 $(\underline{id}) * \underline{id} T'E' \Rightarrow (\underline{id}) * \underline{id} E' \Rightarrow (\underline{id}) * \underline{id} + \underline{TE'} \Rightarrow$   
 $(\underline{id}) * \underline{id} + \underline{FT'E'} \Rightarrow (\underline{id}) * \underline{id} + \underline{id} T'E' \xRightarrow{*} (\underline{id}) * \underline{id} + \underline{id} \$$

$T' \rightarrow \epsilon$

$E' \rightarrow \epsilon$

# FIRST&FOLLOW与预测分析

## M - Table

Consider the following derivation:

What are implications ?

( id ) \* id + id\$ (input)

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow (E) T'E' \Rightarrow$

1.  $M[E, (]$

2.  $M[T, (]$

3.  $M[F, (]$

$(TE') T'E' \Rightarrow (FT'E') T'E' \Rightarrow (id T'E') T'E' \Rightarrow$

$(id E') T'E' \Rightarrow (id) T'E' \Rightarrow (id) * FT'E' \Rightarrow$

4.  $M[E', )]$

$(id) * id T'E' \Rightarrow (id) * id E' \Rightarrow (id) * id + TE' \Rightarrow$

$(id) * id + FT'E' \Rightarrow (id) * id + id T'E' \xrightarrow{*} (id) * id + id\$$

5.  $M[T', \$]$

6.  $M[E', \$]$

1.  $E \rightarrow TE'$  and  $($  in  $\text{First}(TE')$
2.  $T \rightarrow FT'$  and  $($  in  $\text{First}(FT')$
3.  $F \rightarrow (E)$  and  $($  in  $\text{First}('(E)')$
4.  $E' \rightarrow \varepsilon$  and  $)$  in  $\text{Follow}(E')$
5. Since  $\$$  in  $\text{Follow}(T')$ ,  $T' \rightarrow \varepsilon$
6. Since  $\$$  in  $\text{Follow}(E')$ ,  $E' \rightarrow \varepsilon$



# FIRST&FOLLOW的作用

---

## ○ FIRST

- 表示NT（栈）和T（输入流）的关系
- 若 $A \rightarrow \alpha$ ，且 $a \in \text{FIRST}(\alpha)$   
当A在栈顶，输入符号为a  
→ 选择 $A \rightarrow \alpha$ 进行推导——用 $\alpha$ 替换A

# FIRST作用 (续)

栈  
 $\$ \beta A$

输入缓冲区  
 $ay \$$

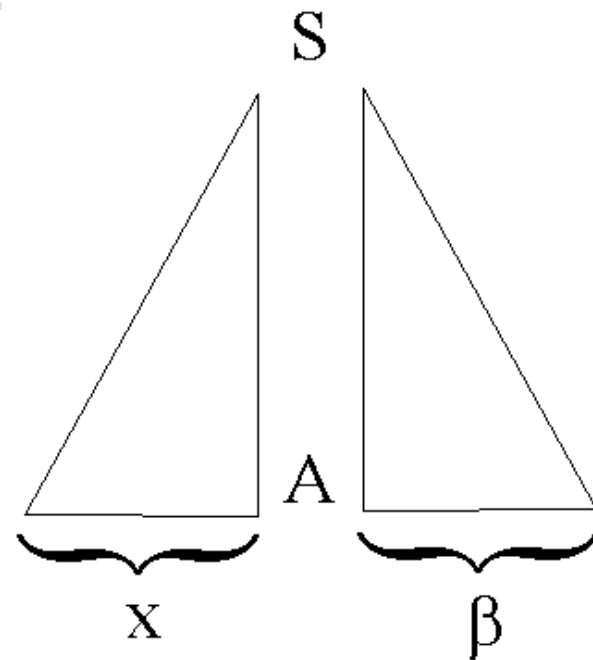
句型:  $x A \beta$

输入:  $x a y$

期待:  $A \beta \Rightarrow \alpha y \overset{*}{\Rightarrow} a y$

即: 想把A变成a...

必然需要 $\alpha$ 的FIRST集合包含a



α为终结符





# FOLLOW的作用

---

## ○ FOLLOW

- 处理FIRST的冲突
- 当 $\alpha = \varepsilon$ 或 $\alpha \xRightarrow{*} \varepsilon$ ，且当前输入符号 $b \in \text{FOLLOW}(A) \rightarrow$  选择 $A \rightarrow \alpha$
- $\alpha$ 最终展开为 $\varepsilon$ ， $b$ 仍为当前符号——“紧接着 $A$ 的符号”。

# FOLLOW的作用 (续)

栈

$\$ \beta A$

句型:  $x A \beta$

A扩展为 $\epsilon$ , 即期待:  $\beta \xRightarrow{*} by$

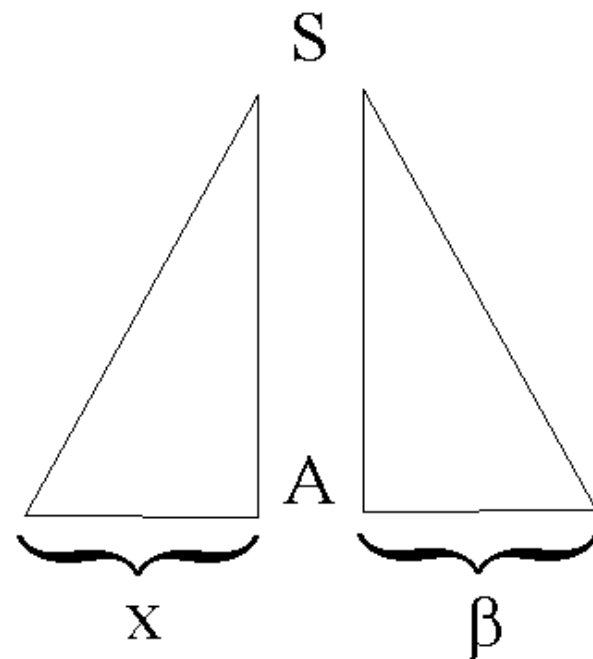
即: 想把A消去, 用 $\beta$ 变成b...

存在句型:  $x A b y$  — FOLLOW

输入缓冲区

$b y \$$

输入:  $x b y$



$A \beta \xRightarrow{*} by$

并期待  $A \Rightarrow \epsilon$

$\beta \xRightarrow{*} by$

$A \Rightarrow \epsilon$

即  $x A b y$  — b Follow A



## 4.4.6 预测分析表的构造

算法4.4

输入：CFG  $G$

输出：预测分析表  $M$

方法：

1. 对每个产生式  $A \rightarrow \alpha$ ，重复做2、3
2. 对所有的终结符  $a \in \text{FIRST}(\alpha)$ ，将  $A \rightarrow \alpha$  加入  $M[A, a]$
3. 若  $\epsilon \in \text{FIRST}(\alpha)$ ：对所有终结符  $b \in \text{FOLLOW}(A)$ ，将  $A \rightarrow \alpha$  加入  $M[A, b]$ ；  
若  $\$ \in \text{FOLLOW}(A)$ ，将  $A \rightarrow \alpha$  加入  $M[A, \$]$
4. 所有未定义的表项设置为错误

# 例4.18

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow FT'$   
 $T' \rightarrow *FT' \mid \epsilon$   
 $F \rightarrow (E) \mid id$

$First(E, F, T) = \{ (, id \}$

$First(E') = \{ +, \epsilon \}$

$First(T') = \{ *, \epsilon \}$

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

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

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

Expression Example:  $E \rightarrow TE' : First(TE') = First(T) = \{ (, id \}$

$M[E, (] : E \rightarrow TE'$

$M[E, id] : E \rightarrow TE'$

规则2

(规则 2)  $E' \rightarrow +TE' : First(+TE') = + : M[E', +] : E' \rightarrow +TE'$

(规则 3)  $E' \rightarrow \epsilon : \epsilon \text{ in } First(\epsilon)$

$M[E', )] : E' \rightarrow \epsilon$  (3.1)

$M[E', \$] : E' \rightarrow \epsilon$  (3.2)

$), \$ \in Follow(E')$

$T' \rightarrow \epsilon : \epsilon \text{ in } First(\epsilon)$

$M[T', +] : T' \rightarrow \epsilon$  (3.1)

$M[T', )] : T' \rightarrow \epsilon$  (3.1)

$M[T', \$] : T' \rightarrow \epsilon$  (3.2)

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

# 例4.19

$S \rightarrow i E t S S' \mid a$	$\text{First}(S) = \{ i, a \}$	$\text{Follow}(S) = \{ e, \$ \}$
$S' \rightarrow eS \mid \epsilon$	$\text{First}(S') = \{ e, \epsilon \}$	$\text{Follow}(S') = \{ e, \$ \}$
$E \rightarrow b$	$\text{First}(E) = \{ b \}$	$\text{Follow}(E) = \{ t \}$

$S \rightarrow i E t S S'$

$S \rightarrow a$

$E \rightarrow b$

$\text{First}(i E t S S') = \{ i \}$

$\text{First}(a) = \{ a \}$

$\text{First}(b) = \{ b \}$

$S' \rightarrow eS$

$S \rightarrow \epsilon$

$\text{First}(eS) = \{ e \}$

$\text{First}(\epsilon) = \{ \epsilon \}$

$\text{Follow}(S') = \{ e, \$ \}$

Non-terminal	INPUT SYMBOL					
	a	b	e	i	t	\$
S	<u><math>S \rightarrow a</math></u>			<u><math>S \rightarrow iEtSS'</math></u>		
S'			<u><math>S' \rightarrow \epsilon</math></u> <u><math>S' \rightarrow eS</math></u>			<u><math>S \rightarrow \epsilon</math></u>
E		<u><math>E \rightarrow b</math></u>				

↑  
2x1t文法

$S \rightarrow i E t S S' | a$

$S' \rightarrow e S | \epsilon$

$E \rightarrow b$

$\text{First}(S) = \{i, a\}$

$\text{First}(S') = \{e, \epsilon\}$

$\text{First}(E) = b$

$\text{Follow}(S) = \{e, \$\}$

$\text{Follow}(S') = \{e, \$\}$

$\text{Follow}(E) = \{t\}$

Non-terminal	INPUT SYMBOL					
	a	b	e	i	t	\$
S	$S \rightarrow a$			$S \rightarrow i E t S S'$		
S'			$S \rightarrow \epsilon$ $S \rightarrow e S$			$S \rightarrow \epsilon$
E		$E \rightarrow b$				



## 4.4.7 LL(1)文法

---

- 例4.19,  $M[S', e]$ 有两个值
- 预测分析表项无多值——LL(1)文法
  - L: 由左至右扫描输入
  - L: 构造最左推导
  - 1: 向前搜索一个输入符号, 结合栈中符号, 即可确定分析器动作



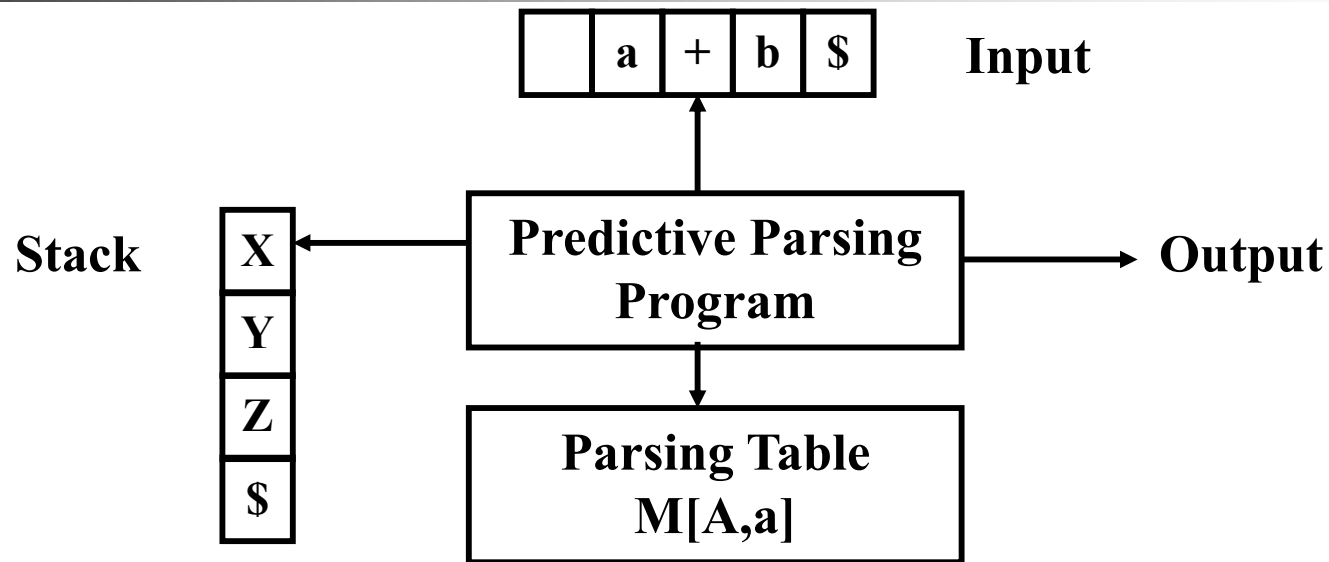
# LL(1)文法的特性

---

1. 无二义性，无左递归
2. 若  $A \rightarrow \alpha \mid \beta$ ，则
  - 1)  $\alpha$ 、 $\beta$ 推导出的符号串，不能以同样的终结符 $a$ 开头
  - 2)  $\alpha$ 、 $\beta$ 至多有一个可推导出 $\varepsilon$
  - 3) 若  $\beta \xRightarrow{*} \varepsilon$ ， $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \Phi$
- 某些语言不存在LL(1)文法，例4.19



## 4.4.8 预测分析法的错误恢复



○ 何时发生错误？

1.  $X \in T$ ,  $X \neq$ 输入符号
2.  $X \in NT$ ,  $M[X, \text{输入符号}]$ 为空

○ 两种策略：Panic模式、短语级恢复



# Panic模式恢复策略

- 考虑NT的同步单词集

1. FOLLOW(A)——略过A
2. 将**高层结构**的开始符号作为**低层结构**的同步集：  
**语句**的开始关键字——**表达式**的同步集，处理赋值语句漏掉分号情况
3. FIRST(A)——重新开始分析A

- 其他方法

4. 若 $A \xRightarrow{*} \epsilon$ ，使用它——推迟错误，减少NT
5. 若T不匹配，可弹出它，报告应插入符号——同步集设置为所有其他单词

# 例4.20

Non-terminal	INPUT SYMBOL					
	id	+	*	(	)	\$
E	$E \rightarrow TE'$	<u>      </u>	<u>      </u>	$E \rightarrow TE'$	<u>sync</u>	<u>sync</u>
E'	<u>      </u>	$E' \rightarrow +TE'$	<u>      </u>	<u>      </u>	$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$	<u>sync</u>	<u>      </u>	$T \rightarrow FT'$	<u>sync</u>	<u>sync</u>
T'	<u>      </u>	$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$	<u>      </u>	$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$	<u>sync</u>	<u>sync</u>	$F \rightarrow (E)$	<u>sync</u>	<u>sync</u>

同步集——FOLLOW集

跳过输入符号，FIRST集

## 例4.20 (续)

STACK	INPUT	Remark
\$E	+ id * + id\$	error, skip + id ∈ FIRST(E)
\$E	id * + id\$	
\$E'T	id * + id\$	
\$E'T'F	id * + id\$	
\$E'T'id	id * + id\$	
\$E'T'	* + id\$	
\$E'T'F*	* + id\$	error, M[F,+] = synch F has been popped
\$E'T'F	+ id\$	
\$E'T'	+ id\$	
\$E'	+ id\$	
\$E'T+	+ id\$	
\$E'T	id\$	
\$E'T'F	id\$	
\$E'T'id	id\$	
\$E'T'	\$	
\$E'	\$	
\$	\$	



# 产生错误信息

---

- 保存输入计数（位置）
- 每个非终结符符号化一个抽象语言结构
- 考虑例4.20文法
  - E表示表达式
    - E在栈顶，输入符号为+：“错误位置i，表达式不能以+开始”或“错误位置i，非法表达式”
    - E，\*的情况类似
  - E'表示表达式的结束
    - E'，\*/id：“错误：位置j开始的表达式在位置i处结构错误”



# 产生错误信息（续）

---

- T表示加法项

- T, \*: “错误位置i, 非法项”

- T'表示项的结束

- T', (: “位置j开始的项在位置i处结构错误”

- F表示加法/乘法项

- 同步错误

- F, +: “位置i缺少加法/乘法项”

- E, ): “位置i缺少表达式”



# 产生错误信息（续）

- 栈顶终结符与输入符号不匹配

- **id, +:** “位置i缺少标识符”

- **)**, 其他符号

- 分析过程中遇到 ‘(’, 都将位置保存在“左括号栈”中——实际可用符号栈实现

- 当发现不匹配时, 查找左括号栈, 恢复括号位置

- “错误位置i: 位置m处左括号无对应右括号”

- 如 **(id \* + (id id)\$**



# 短语级错误恢复

---

- 预测分析表空位填入错误处理函数
  - 修改栈和（或）输入流，插入、删除、替换
  - 输出错误信息
- 问题
  - 插入、替换栈符号应小心，避免错误推导
  - 避免无限循环
- 与Panic模式结合使用，更完整的方式



# 预测分析器的实现

- 栈——容易实现，抽象数据类型
- 输入流——词法分析器实现
- 关键——如何实现预测分析表
- 一种方法：指定唯一ID表示不同表项内容

Non-terminal	INPUT SYMBOL					
	id	+	*	(	)	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	<u><math>T \rightarrow FT'</math></u>	<u>synch</u>		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	<u><math>F \rightarrow id</math></u>	<u>synch</u>	<u>synch</u>	$F \rightarrow (E)$	synch	synch

每个产生式都有唯一的ID

同步动作类似

空白、错误处理

# 修改后的预测分析表

Non-terminal	INPUT SYMBOL					
	id	+	*	(	)	\$
E	1	18	19	1	9	10
E'	20	2	21	22	3	3
T	4	11	23	4	12	13
T'	24	6	5	25	6	6
F	8	14	15	7	16	17

1  $E \rightarrow TE'$       5  $T' \rightarrow *FT'$

2  $E' \rightarrow +TE'$       6  $T' \rightarrow \epsilon$

3  $E' \rightarrow \epsilon$       7  $F \rightarrow (E)$

4  $T \rightarrow FT'$       8  $F \rightarrow id$

9 – 17 :

Sync  
Actions

18 – 25 :

Error  
Handlers



# 修改后的预测分析表（续）

---

- 每个ID（或一组ID）对应一个函数：
  - 使用栈抽象数据类型
  - 获取单词
  - 打印错误信息
  - 打印诊断信息
  - 处理错误



# 程序框架

```
state = M[ top(s), current_token ]
```

```
switch (state)
```

```
{
```

```
case 1: proc_E_TE'();  
        break ;
```

```
...
```

```
case 8: proc_F_id();  
        break ;
```

```
case 9: proc_sync_9();  
        break ;
```

```
...
```

```
case 17: proc_sync_17();  
         break ;
```

```
case 18:
```

```
...
```

```
case 25:
```

```
}
```

调用错误处理函数

组合→使用另一个switch语句

某些同步操作可能是相同的

某些错误处理可能是相似的

# lcc的语法分析——表达式

```
Tree expr(int tok) {  
    static char stop[] = { IF, ID, '}', 0 };  
    Tree p = expr1(0); ← 赋值表达式  
  
    while (t == ',') { ← 表达式序列  
        Tree q;  
        t = gettok();  
        q = pointer(expr1(0));  
        p = tree(RIGHT, q->type, root(value(p)), q);  
    }  
    if (tok)  
        test(tok, stop); ← 错误处理  
    return p;  
}
```

# lcc的语法分析——表达式

```
void skipto(int tok, char set[]) {
```

```
    int n;
```

```
    char *s;
```

```
    assert(set);
```

```
    for (n = 0; t != EOI && t != tok; t = gettok()) {
```

```
        for (s = set; *s && kind[t] != *s; s++)
```

```
            ;
```

```
        if (kind[t] == *s)
```

```
            break;
```

```
        if (n++ == 0)
```

```
            error("skipping");
```

```
        if (n <= 8)
```

```
            printtoken();
```

```
        else if (n == 9)
```

```
            fprintf(stderr, " ...");
```

```
    }
```

其他结构的**FIRST**集



# lcc的语法分析——表达式

---

```
if (n > 8) {  
    fprintf(stderr, " up to");  
    printtoken();  
}  
if (n > 0)  
    fprintf(stderr, "\\n");  
}
```

FOLLOW中的一个单词

```
Tree expr0(int tok) {  
    return root(expr(tok));  
}
```

# 赋值表达式

```
Tree expr1(int tok) {  
    static char stop[] = { IF, ID, 0 };  
    Tree p = expr2();
```

条件表达式

```
    if (t == '='  
        || (prec[t] >= 6 && prec[t] <= 8)  
        || (prec[t] >= 11 && prec[t] <= 13)) {  
        int op = t;  
        t = gettok();  
        if (oper[op] == ASGN)  
            p = asgntree(ASGN, p, value(expr1(0)));  
        else
```

多重赋值



# 赋值表达式

```
{
    expect('=');
    p = incr(op, p, expr1(0));
}

if (tok)
    test(tok, stop);
return p;
}
```

根据不同的运算符创建相应的语法树

```
Tree incr(int op, Tree v, Tree e) {
    return asgntree(ASGN, v, (*optree[op])(oper[op], v, e));
}
```

“运算” 赋值

# 二元表达式

运算符（表达式）优先级

```
void expr3(int k) {  
    if (k > 13)  
        unary();  
    else {  
        expr3(k + 1);  
        while (pret[t] == k) {  
            t = gettok();  
            expr3(k + 1);  
        }  
    }  
}
```

不妨： $k \text{——} +、- , k+1 \text{——} *、/$

加减法表达式的结构——乘除法表达式（**term**）通过 $+$ 、 $-$ 连接的序列



# 消除递归

---

```
void expr3(int k) {  
    int k1;  
    unary();  
    for (k1 = prec[t]; k1 >= k; k1--)  
        while (pret[t] == k1) {  
            t = gettok();  
            expr3(k1 + 1);  
        }  
}
```



# 自顶向下分析——总结

---

- 已经研究了上下文无关文法、语言理论及与语法分析的关系
- 关键：重写文法适应分析方法需求
- 自顶向下分析方法
  - 平凡方法：状态转换图 & 递归
  - 高效方法：表驱动
- 缺点：不是所有文法满足LL(1)要求