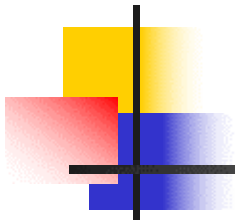


第四章 语法分析



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

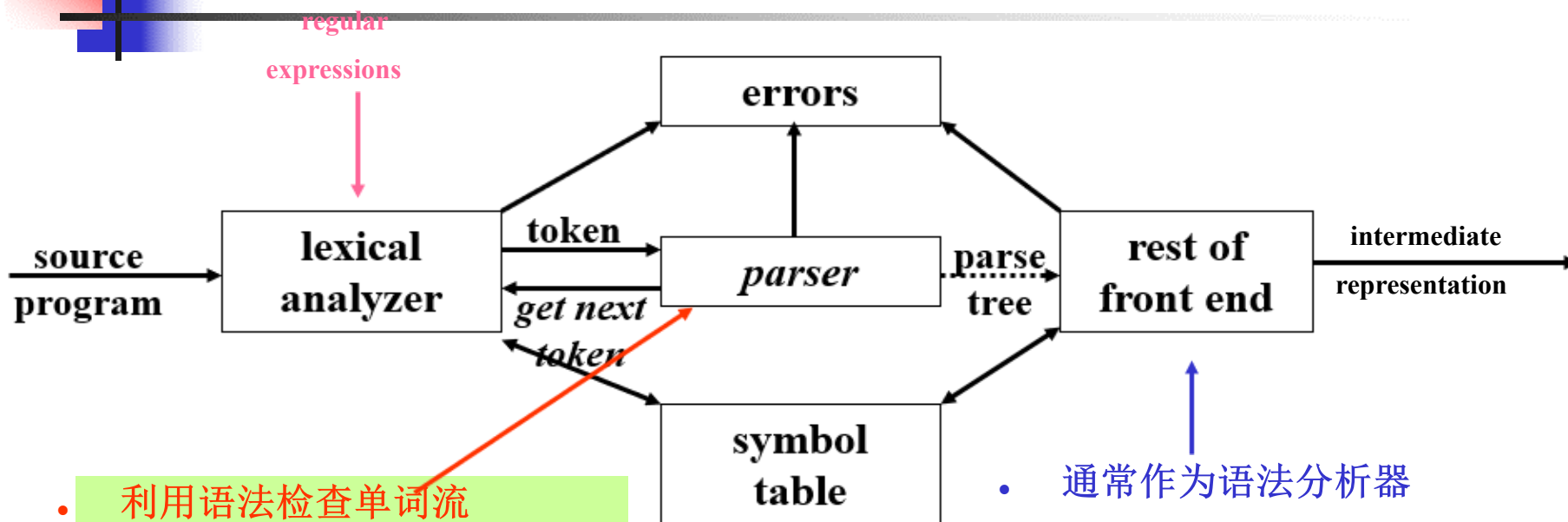
- 基于语法的翻译方法描述——□程序^{工具}

- 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

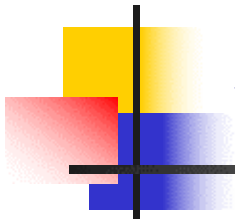
- ADA □ ADA9x, C++增加: 模板、异常**

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



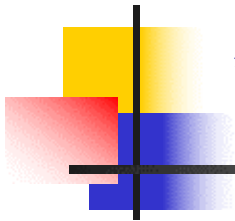
三类语法分析器

- 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



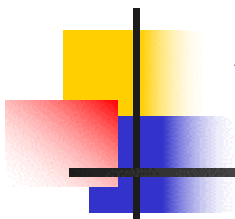
4.1.1 语法错误处理

- 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
- “,” □ “;”，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

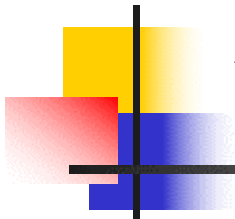


4.2 上下文无关文法

○ 定义：四元式(V_T, V_N, S, 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. 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

$expr \rightarrow expr\ op\ expr$

$expr \rightarrow (expr)$

$expr \rightarrow -\ expr$

$expr \rightarrow id$

$op \rightarrow +$

$op \rightarrow -$

$op \rightarrow *$

$op \rightarrow /$

$op \rightarrow \square$

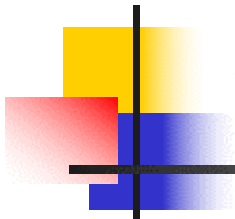
蓝色符号——T，黑色符号——NT



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

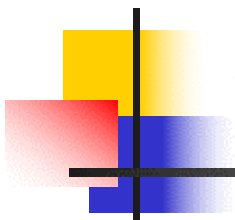
- 字母表靠前的大写字母, A、B、C ...
- S, 通常作为开始符号
- 斜体小写字母串, *expr*、*term*、...

3. 字母表靠后的大写字母

- 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^*



符号约定（续）

5. 小写希腊字母

□ α 、 β 、 γ ..., 语法符号串, $(T \cup NT)^*$

6. 左部相同的产生式可合并, ‘|’ —— “或”

□ $A \rightarrow \alpha_1; A \rightarrow \alpha_2; \dots; A \rightarrow \alpha_k;$

□ $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$, 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

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



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）

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

○ $E \square -E$, E 可替换为 $-E$

$E \square -E$, “ E **直接推出** $-E$ ”

$E * E \square (E) * E$

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

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

定义

○ 形式化定义

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

□ $\alpha_1 \alpha_2 \dots \alpha_n \xRightarrow{*} \alpha_1 \alpha_n$

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

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

$\xRightarrow{*}$, “一步或多步推导”, 推导步数 ≥ 1

$\xRightarrow{+}$, “0步或多步推导”, 推导步数 ≥ 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

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

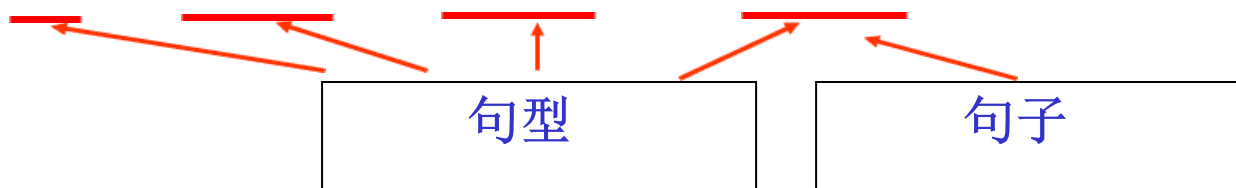
- w: G的一个**句子**, sentence
- CFG生成**上下文无关语言**
- 两个CFG生成相同语言，两个CFG**等价**
- $S \Rightarrow^* \alpha$, α 可能包含NT

α : G的一个**句型**, **sentential form**

句子: 不包含NT的句型

例4.4

$E \Rightarrow E * E \Rightarrow id * E \Rightarrow id * id$



$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(id+E) \Rightarrow -(id+id)$

另一种推导过程

$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(E+id) \Rightarrow -(id+id)$

最左推导和最右推导

- 最左推导：总替换最左边的NT

$$E \xrightarrow{lm} -E \xrightarrow{lm} -(E) \xrightarrow{lm} -(E+E) \xrightarrow{lm} -(id+E) \xrightarrow{lm} -(id+id) \quad lm$$

- 最右推导：总替换最右边的NT

$$E \xrightarrow{rm} -E \xrightarrow{rm} -(E) \xrightarrow{rm} -(E+E) \xrightarrow{rm} -(E+id) \xrightarrow{rm} -(id+id) \quad rm$$

- 形式化定义： $A \Rightarrow \alpha$

$$wA \Rightarrow w\alpha, w \text{ 只含 } T$$

$$\alpha Aw \Rightarrow \alpha\beta w, w \text{ 只含 } T$$

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



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□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□句型，**yield**, **frontier**

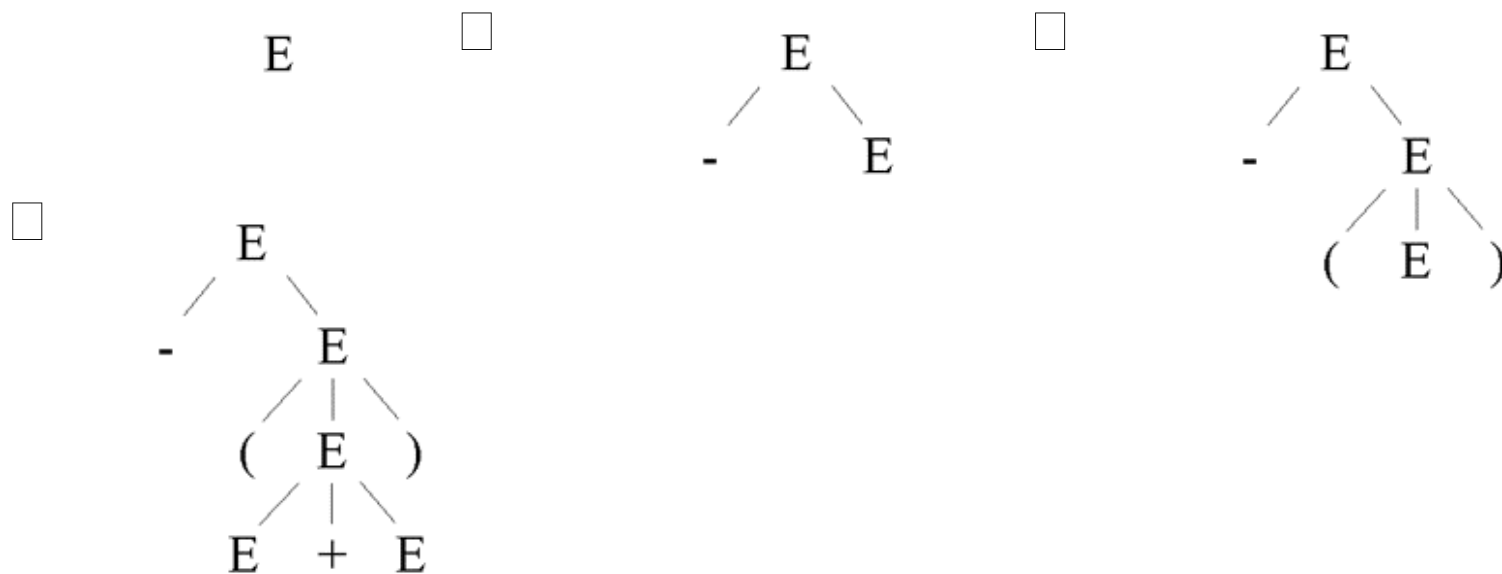


语法树与推导的关系

- 一个推导过程: $\square_1 \square_2 \cdots \square_n$
 - $\square_1 \equiv A$, 单节点, 标记为A
 - $\square_{i-1} = X_1 X_2 \cdots X_k$ 对应语法树T
 - 第i步推导, $X_j \rightarrow Y_1 Y_2 \cdots Y_r$
 - T的第j个叶节点, 添加r个孩子节点 Y_1, Y_2, \cdots, Y_r , 特殊情况, $r=0$, 一个孩子 ε

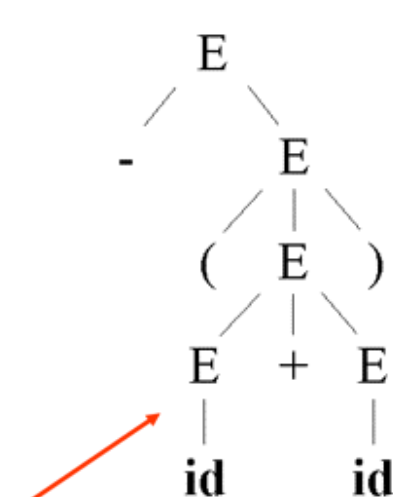
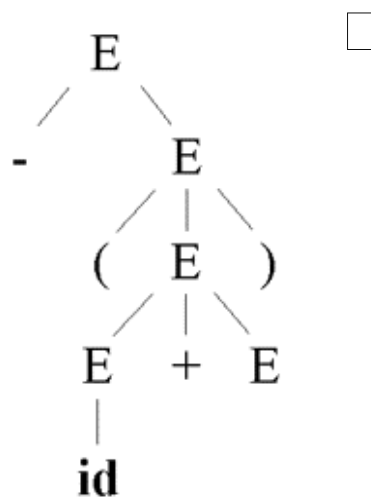
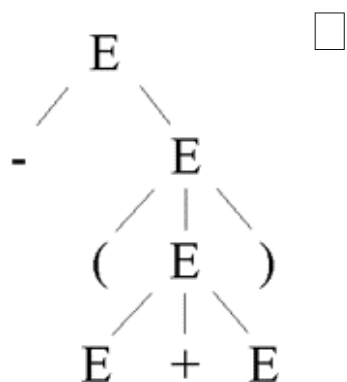
例4.5

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



例4.5

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



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

一棵语法树可 \Rightarrow 多个推导

一棵语法树 \Rightarrow 唯一最左推导，唯一最右推导

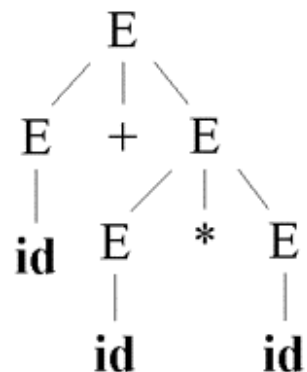
4.2.4 二义性文法（例4.6）

$E \rightarrow E+E \mid \text{id}+E$

$\mid \text{id}+E^*E$

$\mid \text{id}+\text{id}^*E$

$\mid \text{id}+\text{id}^*\text{id}$

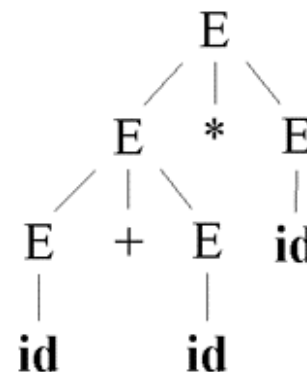


$E \rightarrow E^*E \mid E+E^*E$

$\mid \text{id}+E^*E$

$\mid \text{id}+\text{id}^*E$

$\mid \text{id}+\text{id}^*\text{id}$



○ 句子 $\text{id}+\text{id}^*\text{id}$ 多个语法树，多个最左（右）推导



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,

$(a|b)^*abb$

$A_0 \sqsubseteq aA_0 \mid aA_1 \mid bA_0$

$A_1 \sqsubseteq bA_2$

$A_2 \sqsubseteq bA_3$

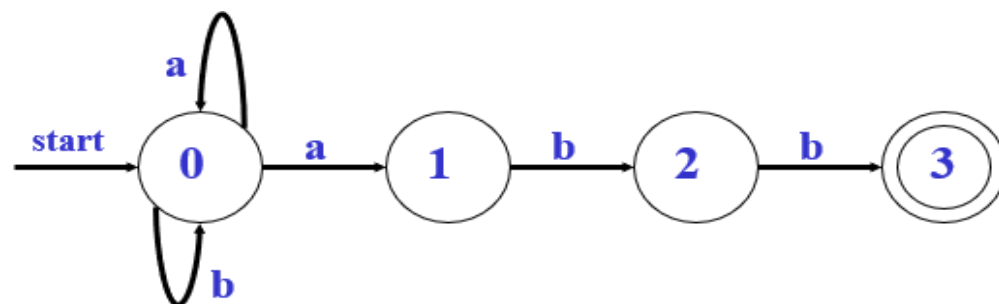
$A_3 \sqsubseteq \varepsilon$

- 正规语言 \sqsubseteq 上下文无关语言

Reg. Lang.

CFLs

NFA \square CFG



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

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

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

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

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

5. 若 i 为初态, A_i 为开始符号: A_0

○ 正则文法



NFA \square CFG

- A_i 的含义是什么？状态 $i \square$ 终态路径上的符号串集合
- A_i 取“初态 \square 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？

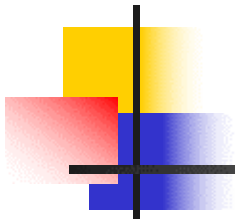
- $A_j \square A_i a$

$$A_0 \square \epsilon$$

$$A_0 \square A_0 a, A_1 \square A_0 a$$

$$A_0 \square A_0 b, A_2 \square A_1 b$$

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



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 $L = \{ \text{所有括号组成的, 且括号匹配的字符串, 且只有这些字符串} \}$



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， ε ，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

$$S \Rightarrow (S)S \Rightarrow (\overset{*}{x})S \Rightarrow (x)y^*$$

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

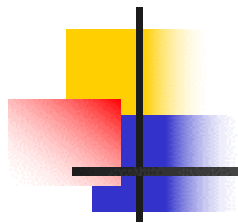
长度 $=2n$ 的符号串 w 。

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

$$S \Rightarrow (S)S \Rightarrow (x)S^* \Rightarrow (x)y^*$$

由1、2，二得证

由一、二，原命题得证

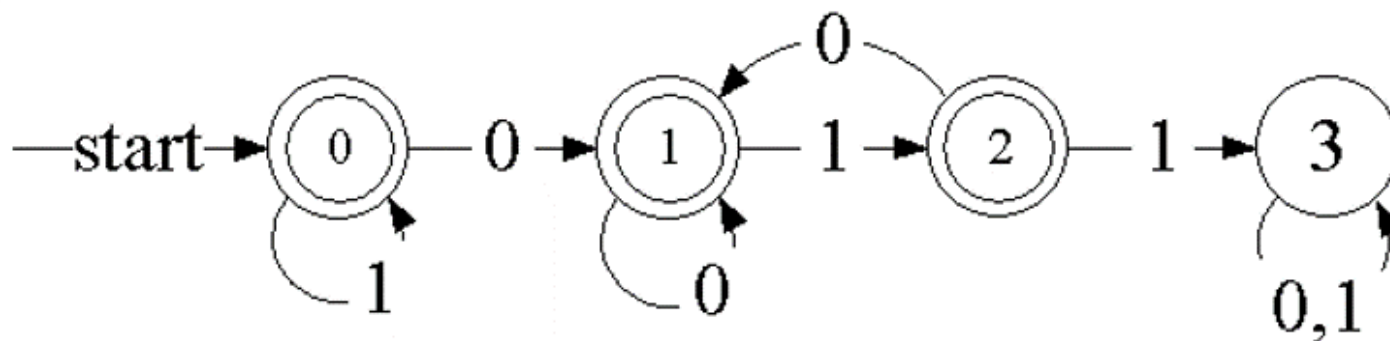


设计CFG练习

- 基本的递归
- $L = \{ a^n b b^{2n} \mid n \geq 0 \}$

$S \rightarrow b \mid aSbb$

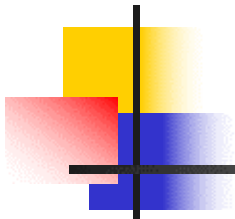
不包含子串011的0/1串


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



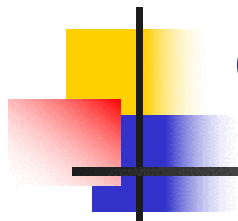
形如 $xy(x \neq y)$ 的01串

- 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?



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
- ϵ -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

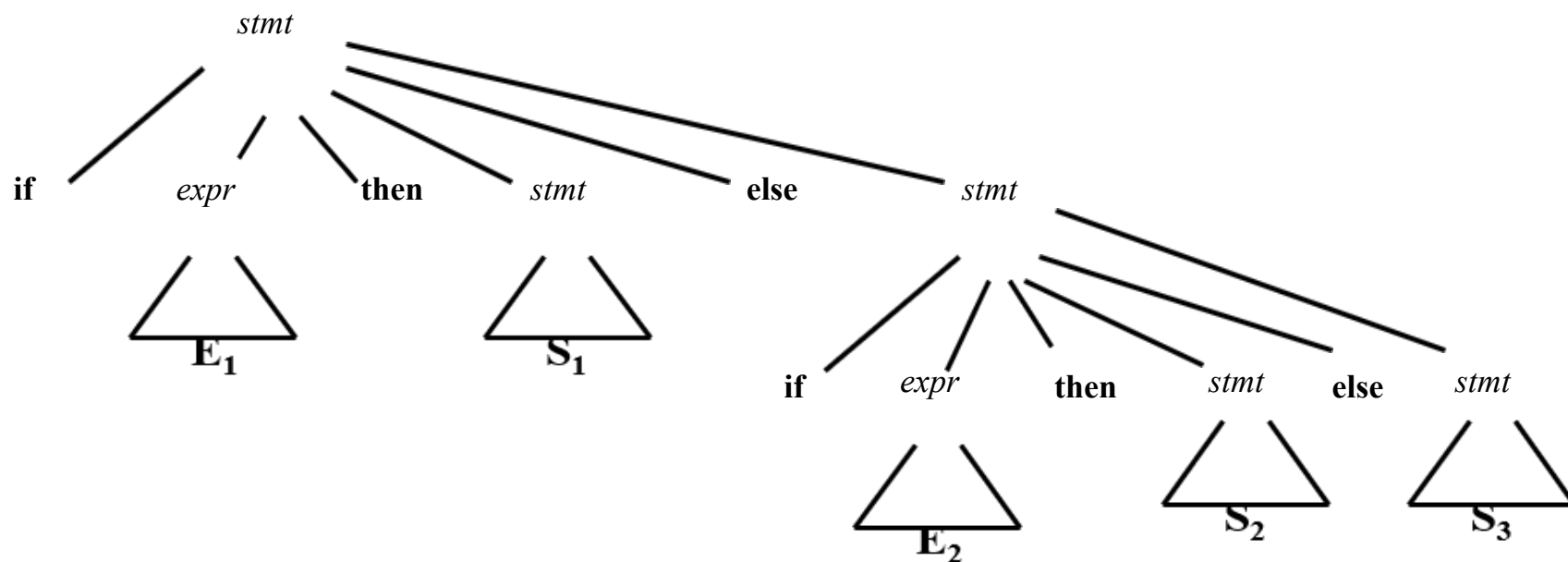
stmt \square **if** *expr* **then** *stmt*

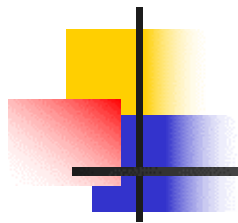
| **if** *expr* **then** *stmt* **else** *stmt*

| **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

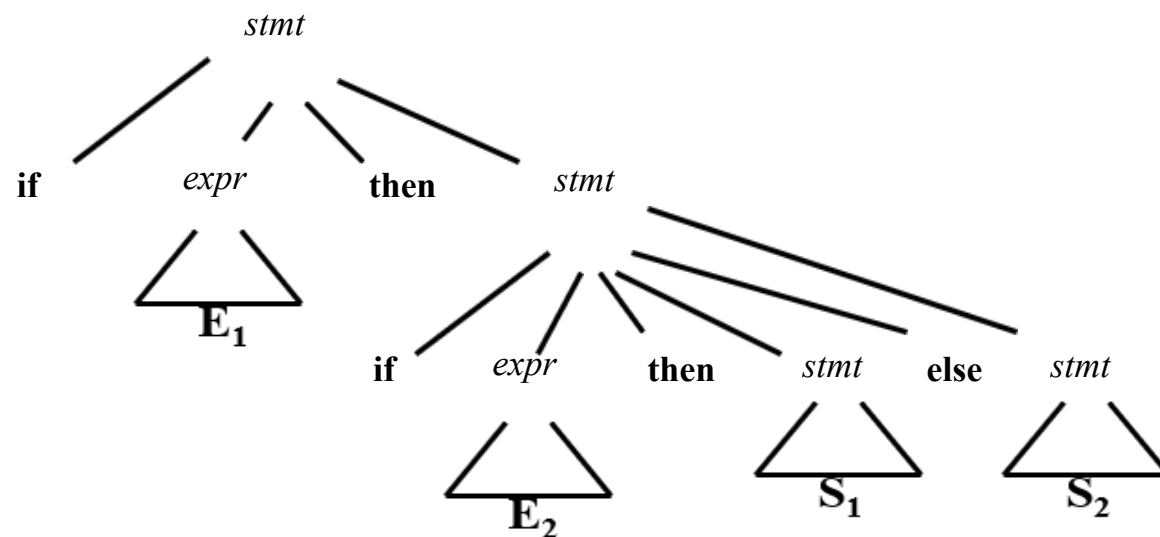
if E_1 then
 if E_2 then
 S_1
 else
 S_2

vs.

if E_1 then
 if E_2 then
 S_1
else
 S_2

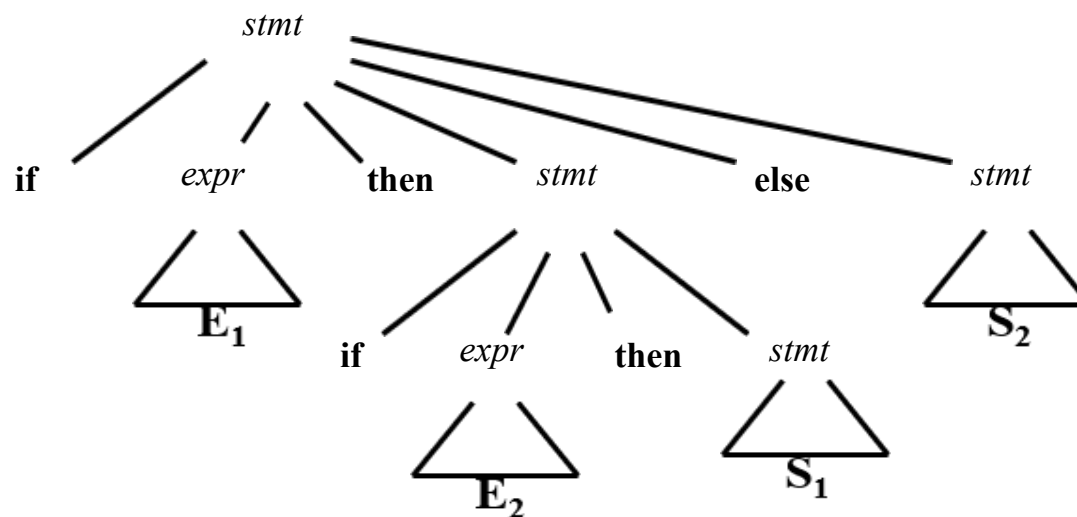
两个语法树

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

stmt \square *matched_stmt*

| *unmatched_stmt*

matched_stmt \square **if** *expr* **then** *matched_stmt* **else** *matched_stmt*

| **other**

unmatched_stmt \square **if** *expr* **then** *stmt*

| **if** *expr* **then** *matched_stmt* **else** *unmatched_stmt*



4.3.4 消除左递归

- $A \rightarrow^+ A\alpha$
- 自顶向下分析方法无法处理，死循环
- 直接左递归的消除

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$$

任何 α_i 均不以A开头，改写为：

$$A \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_n A'$$

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



例4.8

$E \square E + T \mid T$

$\longrightarrow \begin{cases} E \square TE' \\ E' \square + TE' \mid \varepsilon \end{cases}$

$T \square T * F \mid F$

$\longrightarrow \begin{cases} T \square FT' \\ T' \square * FT' \mid \varepsilon \end{cases}$

$F \square (E) \mid id$

$\longrightarrow F \square (E) \mid id$



算法4.1：消除间接左递归

输入：CFG G ，无环路，无 ϵ 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

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

1. 间接左递归 \square 直接左递归:

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

2. 消除直接左递归

$$S \sqsupset Aa \mid b$$
$$A \sqsupset bdA' \mid A'$$
$$A' \sqsupset cA' \mid adA' \mid \varepsilon$$

补充：消除 ε 产生式

○ 方法：利用产生式进行代入

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

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

$E \rightarrow TE' \mid T$

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


$T \rightarrow FT' \mid F$

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

$F \rightarrow (E) \mid id$



补充：消除 ε 产生式（续）

$$A_1 \sqsupset A_2 a \mid b$$
$$A_2 \sqsupset bd A_2' \mid A_2'$$
$$A_2' \sqsupset c A_2' \mid bd A_2' \mid \varepsilon$$

$$\begin{aligned} A_1 &\sqsupset A_2 a \mid b \mid a \\ A_2 &\sqsupset bd A_2' \mid A_2' \\ &\quad \mid bd \\ A_2' &\sqsupset c A_2' \mid bd A_2' \\ &\quad \mid c \mid bd \end{aligned}$$



补充：消除回路

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

回路: $S \square SS \square S$

$S \square \varepsilon$

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

$S \square S(S) \mid (S) \mid \varepsilon$



4.3.5 提取左公因子

- 预测分析方法要求——

向前搜索一个单词，即可确定产生式

- $stmt \sqsubseteq \mathbf{if\ expr\ then\ stmt\ else\ stmt}$

| $\mathbf{if\ expr\ then\ stmt}$ 不符合!

- 一般的

$A \sqsubseteq \square \square \square_1 \mid \square \square_2$

改写为

$A \sqsubseteq \square \square A'$

$A' \sqsubseteq \square \square_1 \mid \square_2$



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 α ，若 $\alpha \neq \varepsilon$ ，则将所有 A 的候选式

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

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

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



例4.10

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

$i \sqsupset \mathbf{if}$, $t \sqsupset \mathbf{then}$, $e \sqsupset \mathbf{else}$, $E \sqsupset$ 表达式, $S \sqsupset$ 语句

改写为:

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



4.3.6 CFG无法描述的语言结构

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

- 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



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

- $L_1' = \{ wcw^R \mid w \in (a \mid b)^*, w^R \text{为} w \text{的反转} \}$

$S \sqsubseteq aSa \mid bSb \mid c$

- $L_2' = \{ a^n b^m c^m d^n \mid n \geq 1 \text{ 且 } m \geq 1 \}$

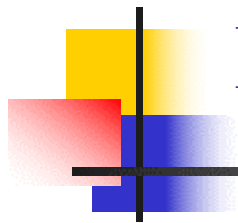
$S \sqsubseteq aSd \mid aAd \quad A \sqsubseteq bAc \mid bc$

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

$S \sqsubseteq AB \quad A \sqsubseteq aAb \mid ab \quad B \sqsubseteq cBd \mid cd$

- $L_3' = \{ a^n b^n \mid n \geq 0 \}$

$S \sqsubseteq aSb \mid ab$



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

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

总状态数 $k \leq s_0, s_1, \dots, s_k$ 中至少有两个相同状态,

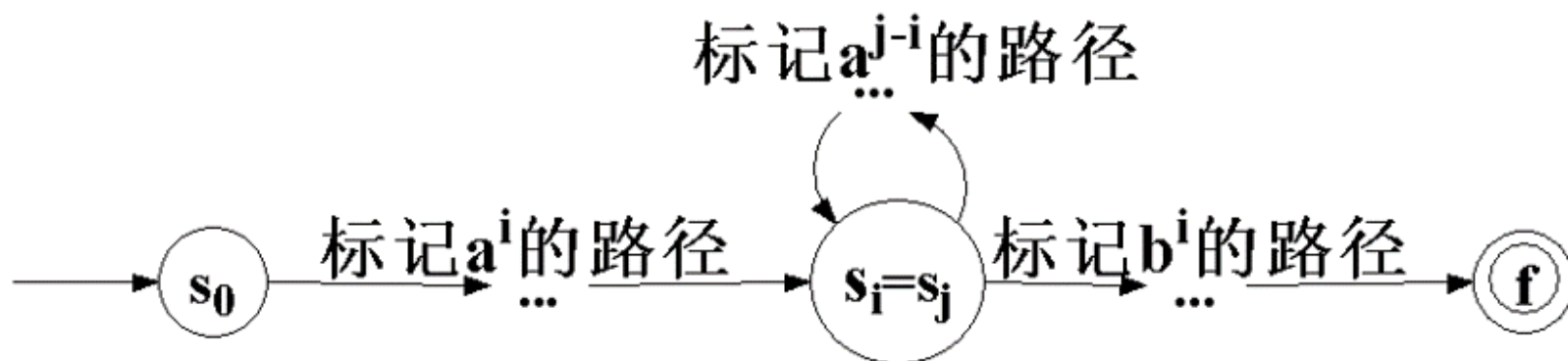
不妨设为 $s_i, s_j, i < j$

L_3' 用正规式无法描述 (续)

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

\square 初态 \square 终态还有标为 $a^j b^i$ 的路径 $\square D$ 接受 $a^j b^i$,

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

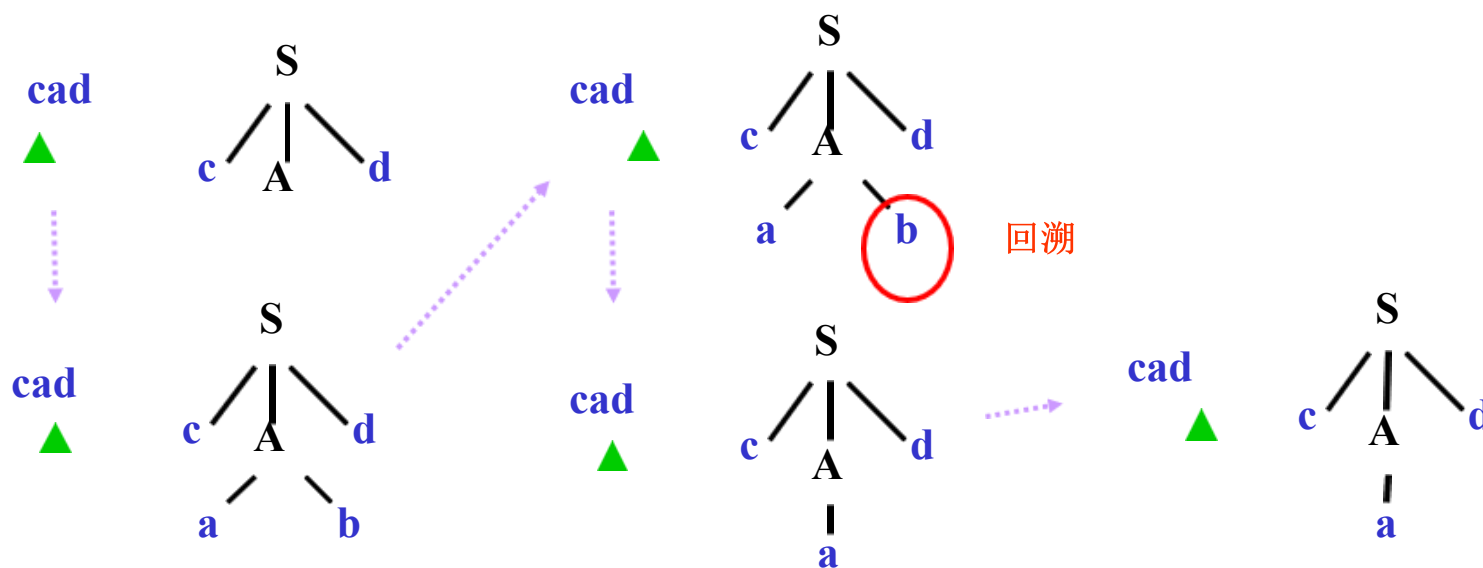
- 自顶向下分析方法的一般策略
- 根据输入符号选择产生式
- 选择错误，需要回溯
- 分析程序语言结构，回溯很少发生

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
- “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 □ 状态转换图

- 边的标记

- 单词：与下一个输入符号匹配 □ 状态转换

- 非终结符：对其进行扩展（调用对应函数）

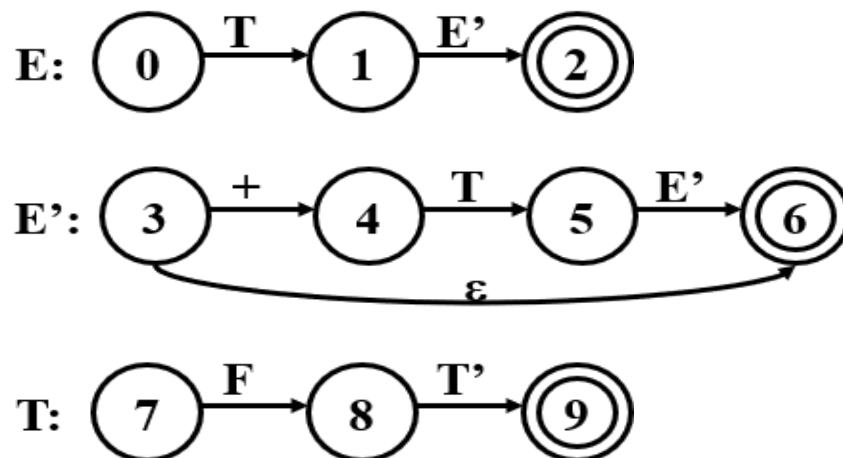
- 创建NT A对应TD:

- 1. 创建一个初态和一个终态

- 2. 对每个产生式 $A \rightarrow X_1 X_2 \cdots X_n$ ，创建初态到终态的一条路径，边标记为 X_1, X_2, \cdots, 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？

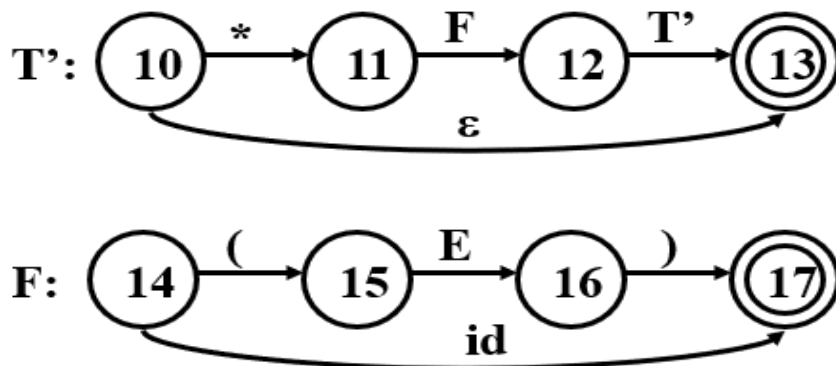
ε 边如何处理？

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:

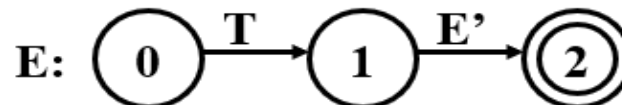
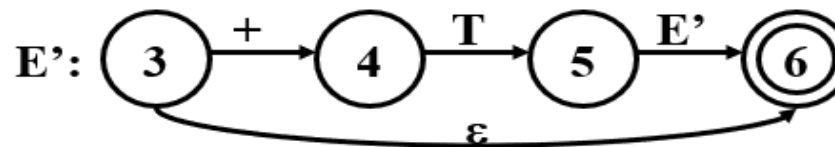
1. TD当前状态为s, 下个输入符号为a, 存在边 $s \xrightarrow{a} t$ 输入指针前移, 当前状态 $\rightarrow t$
2. 存在边 $s \xrightarrow{NTA} t$ 当前状态 $\rightarrow A$ 的TD的初态, 输入指针不变。当到达A的终态, 立即转换到 t ——读入“A”使得从s转换到t
3. 存在边 $s \xrightarrow{\epsilon} t$ 当前状态 $\rightarrow t$, 输入指针不变。

使用TD（代码示例）

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

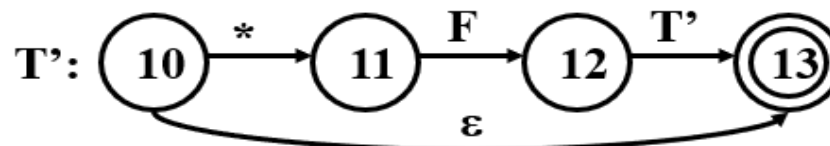
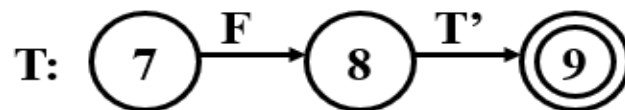
```
TD_E()
{
    TD_T();
    TD_E'();
}
```

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



使用TD（代码示例）

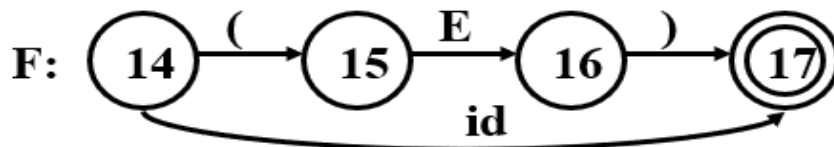
```
TD_T()  
{  
    TD_F();  
    TD_T'();  
}
```



```
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
      ... }
```



ϵ 边如何处理?

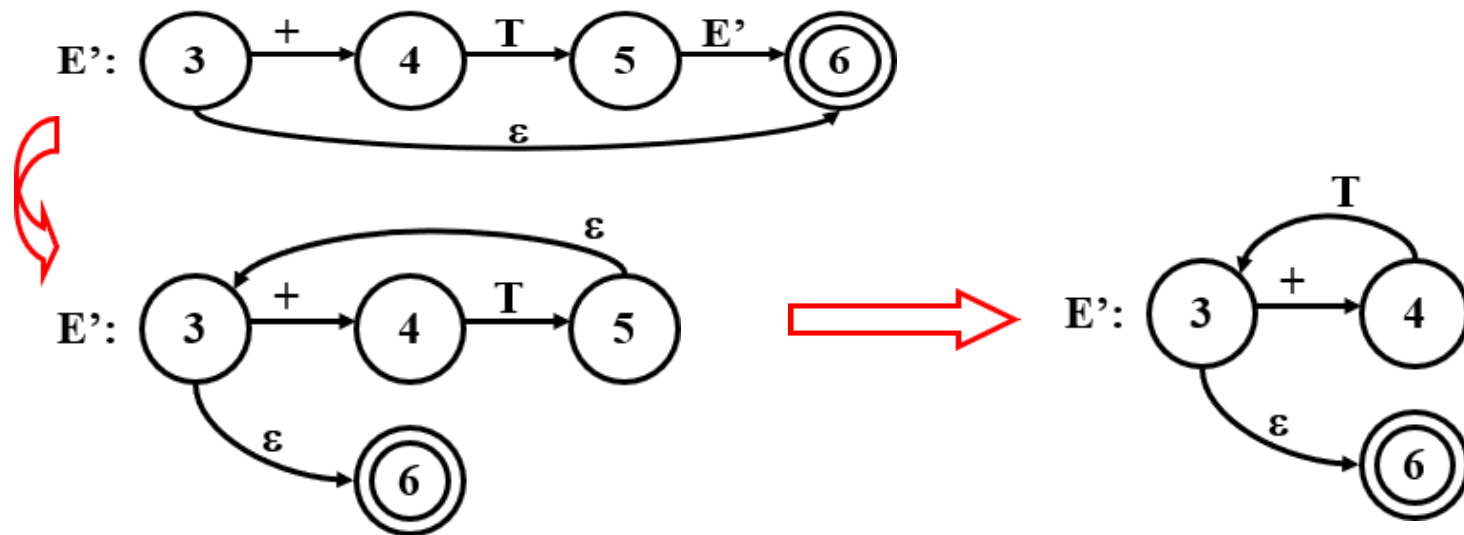
... “else unget()

and terminate”

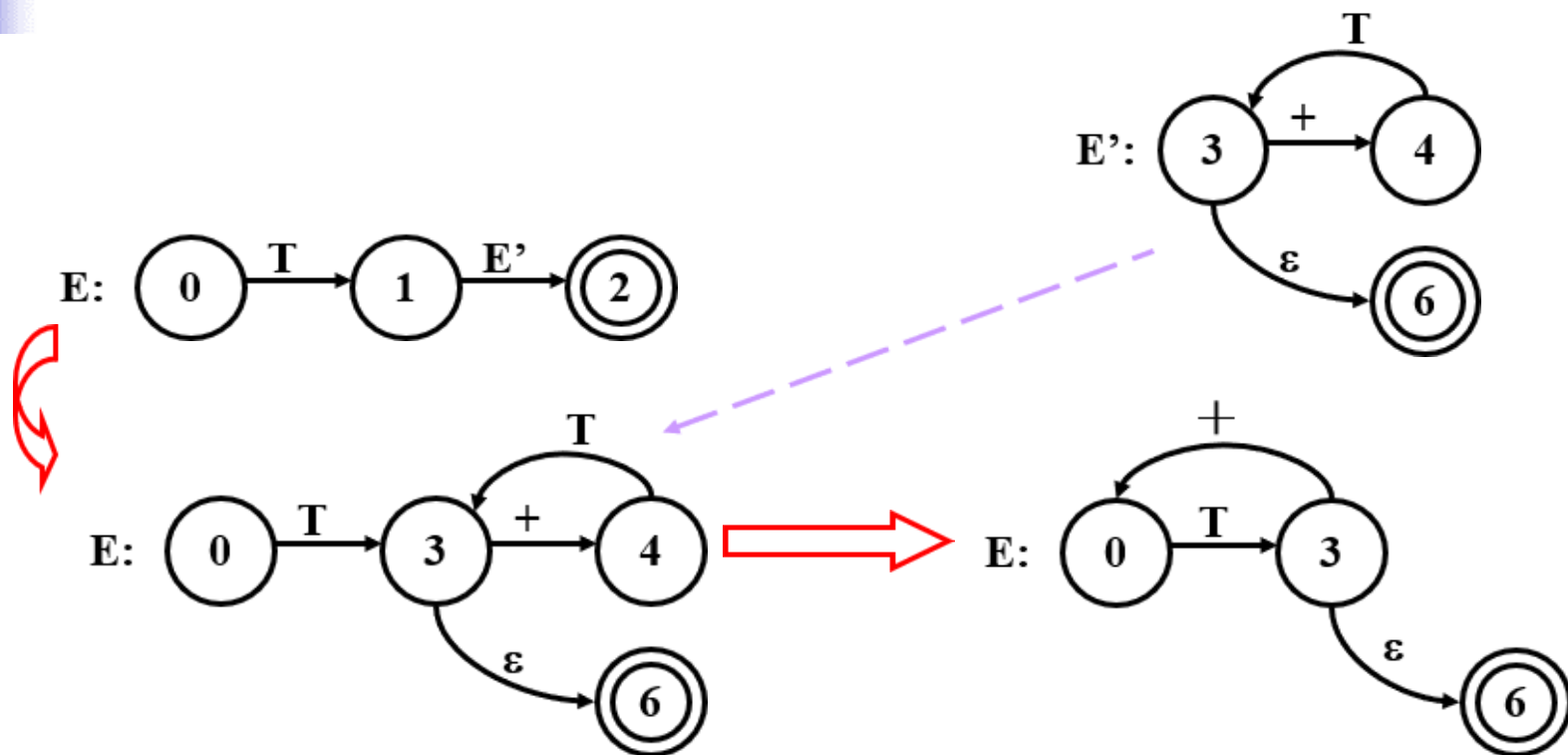
NOTE: 并未给

出所有错误的处
理代码

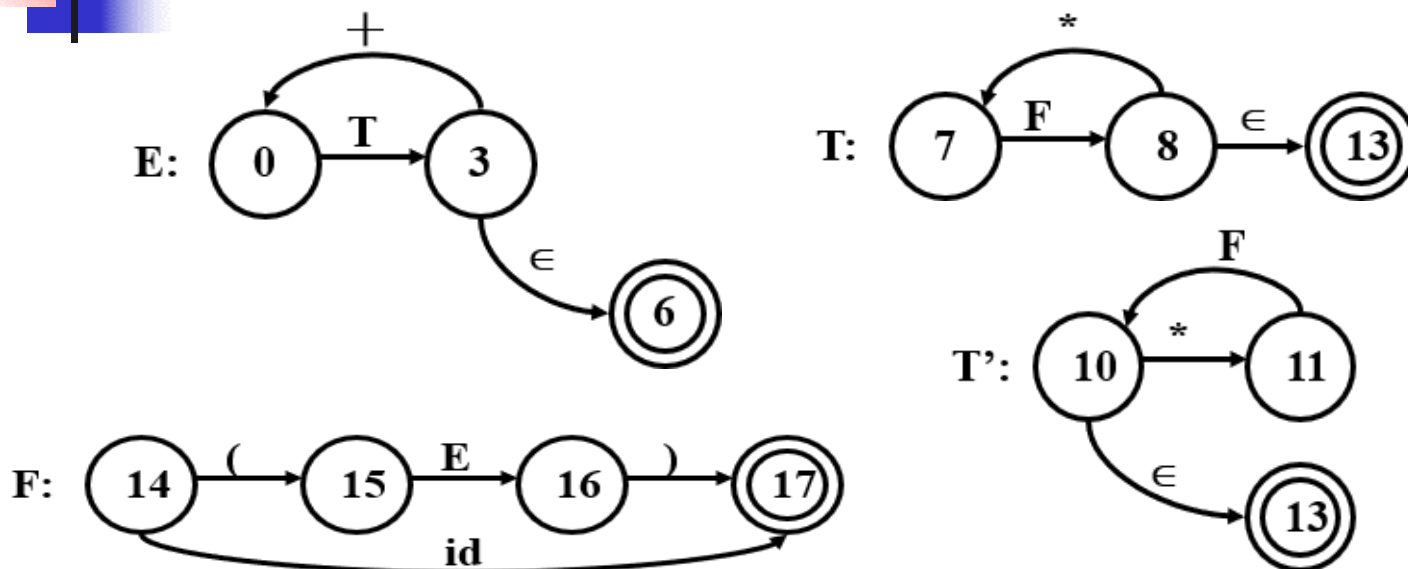
TD的化简



TD的化简

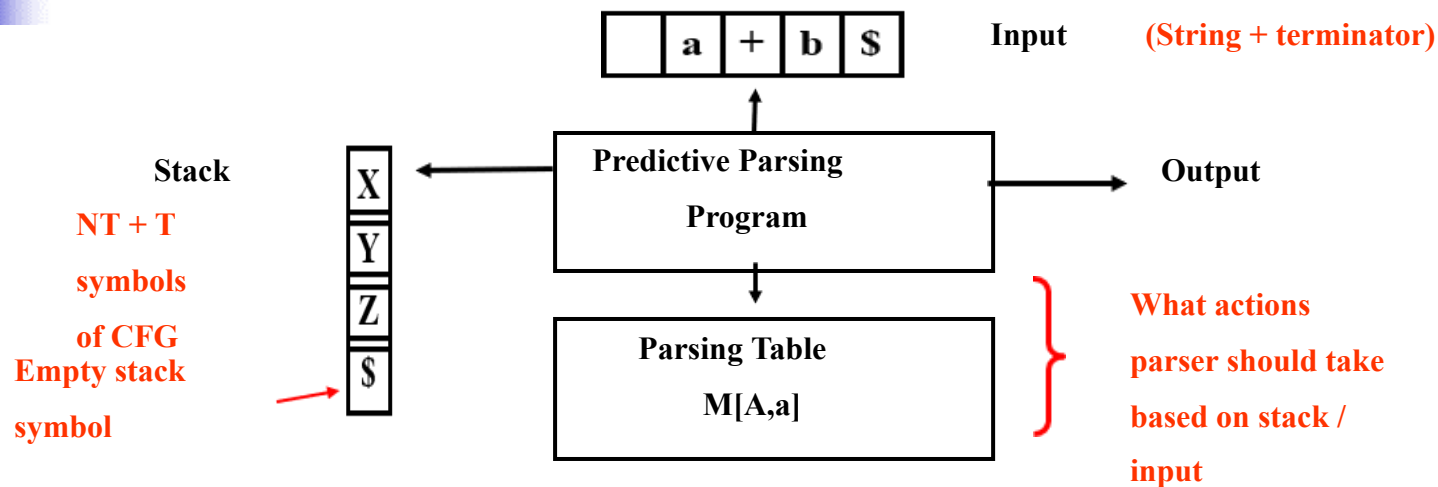


完整的化简



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



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

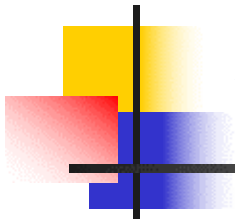
析器运行算法：

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  $\square$  Y1Y2...Yk) {  
    X弹出栈;  
    将Yk, Yk-1, ..., Y1压栈，Y1置于栈顶;  
    输出产生式X  $\square$  Y1Y2...Yk;  
} 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
SE	id + id * id\$	
SE'T	id + id * id\$	$E \rightarrow TE'$
SE'T'F	id + id * id\$	$T \rightarrow FT'$
SE'T'id	id + id * id\$	$F \rightarrow id$
SE'T'	+ id * id\$	
SE'	+ id * id\$	$T' \rightarrow \epsilon$
SE'T+	+ id * id\$	$E' \rightarrow \underline{+}TE'$
SE'T	id * id\$	
SE'T'F	id * id\$	$T \rightarrow FT'$
SE'T'id	id * id\$	$F \rightarrow id$
SE'T'	* id\$	
SE'T'F*	* id\$	$T' \rightarrow \underline{*}FT'$
SE'T'F	id\$	
SE'T'id	id\$	$F \rightarrow id$
SE'T'	\$	
SE'	\$	$T' \rightarrow \epsilon$
\$	\$	$E' \rightarrow \epsilon$

Expend Input



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？

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

- 所有 α 可推导出的符号串的开头终结符的集合

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

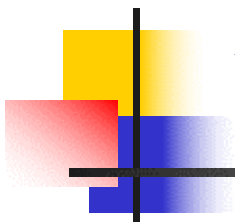


FIRST和FOLLOW

○ FOLLOW?

□ FOLLOW(A): $A \in NT$

- 所有句型中紧接A之后的终结符的集合
- $S^* \square A a \square \square a \in FOLLOW(A)$
- $S^* \square A \square \$ \in FOLLOW(A)$



计算单个符号的FIRST函数

1. 若 X 是终结符, 则 $\text{FIRST}(X)=\{X\}$
2. 若 $X \sqsupseteq \varepsilon$, 则将 ε 加入 $\text{FIRST}(X)$
3. 若 $X \sqsupseteq Y_1 Y_2 \cdots Y_k$, 则

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

若 $Y_1 \sqsupseteq \varepsilon$ $\text{FIRST}(Y_2)$ 加入 $\text{FIRST}(X)$

若 $Y_2 \sqsupseteq \varepsilon$ $\text{FIRST}(Y_3)$ 加入 $\text{FIRST}(X)$

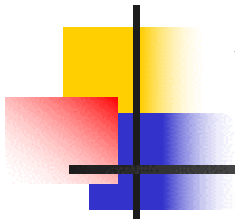
...

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

若 $Y_k \sqsupseteq \varepsilon$ ε 加入 $\text{FIRST}(X)$

NOTE: 一旦 $Y_i \sqsupseteq \varepsilon$, 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

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

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

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

...

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

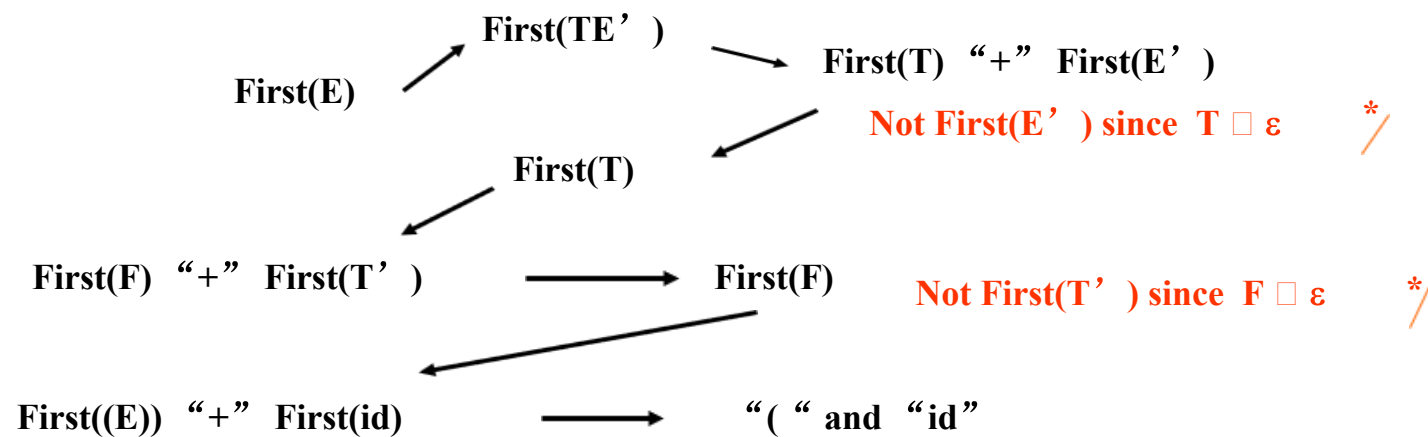
注意：仅当对所有 i , $\varepsilon \in \text{FIRST}(X_i)$, 才将 ε 加入

$\text{FIRST}(X_1 X_2 \cdots X_n)$

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

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

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



Given the production rules:

$$S \rightarrow i E \mid SS' \mid a$$
$$S' \rightarrow eS \mid \varepsilon$$
$$E \rightarrow b$$

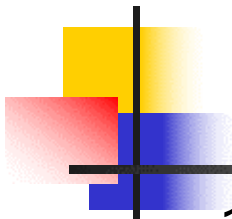
Verify that

$$\text{First}(S) = \{ i, a \}$$
$$\text{First}(S') = \{ e, \varepsilon \}$$
$$\text{First}(E) = \{ b \}$$



计算FOLLOW(A)——非终结符

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



二次扫描算法计算FOLLOW

1. 对所有非终结符 X ， $FOLLOW(X)$ 置为空集。

$FOLLOW(S) = \{\$ \}$ ， 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 $X \rightarrow X_1 X_2 \cdots X_m$

for $j = 1$ to m

if X_j 为非终结符，则

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

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

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

例4.17

Compute Follow for:

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

FIRST

$E \rightarrow (id$

$E' \rightarrow \varepsilon +$

$T \rightarrow (id$

$T' \rightarrow \varepsilon *$

$F \rightarrow (id$

FOLLOW

$E \rightarrow \$)$

$E' \rightarrow \$)$

$T \rightarrow + \$)$

$T' \rightarrow + \$)$

$F \rightarrow + * \$)$

例

Recall:

$S \sqsubseteq iEtSS' \mid a$

$S' \sqsubseteq eS \mid \varepsilon$

$E \sqsubseteq b$

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

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

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

FOLLOW(S) - 包含\$, S是开始符号

$S \sqsubseteq iEtSS'$, 将**FIRST(S')**加入 - 除去 ε

$S' \sqsubseteq \varepsilon$, 将**FOLLOW(S)**加入

$S' \sqsubseteq eS$, 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?

$\text{First}(E) = \{ (, id \}$

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

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

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

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

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

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

FIRST与预测分析

Consider the following derivation:

What's First for each non-terminal?

First(T) = { (, id }

id, (∈ First(F)

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

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

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

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

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

FIRST与预测分析

Consider the following derivation:

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

What's First for each non-terminal?

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

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

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

$T' \Rightarrow \epsilon$

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

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

$T' \Rightarrow \epsilon$

FIRST与预测分析

Consider the following derivation:

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

What's First for each non-terminal?

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

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

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

$$E' \sqsubseteq \varepsilon$$

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

=

$$(\text{id}) * \text{id} + FT' E' \sqsubseteq (\text{id}) * \text{id} + \text{id } T' E' \sqsubseteq (\text{id}) * \text{id}^* + \text{id} \$$$

$$E' \sqsubseteq \varepsilon$$

FIRST与预测分析

Consider the following derivation:

First(F) = { (, id }

What's First for each non-terminal ?

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

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

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

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

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

FIRST与预测分析

Consider the following derivation:

What's First for each non-terminal ?

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

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

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

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

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

FOLLOW与预测分析

Consider the following derivation:

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

What's Follow for each non-terminal?

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

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

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

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

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

FOLLOW与预测分析

Consider the following derivation:

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

What's Follow for each non-terminal?

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

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

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

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

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

FOLLOW与预测分析

Consider the following derivation:

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

What's Follow for each non-terminal?

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

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

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

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

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

FOLLOW与预测分析

Consider the following derivation:

What's Follow for each non-terminal?

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

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

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

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

$T' \Rightarrow \epsilon$

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

$E' \Rightarrow \epsilon$

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

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

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

$$\begin{aligned}
 E &\sqsubseteq TE' \sqsubseteq FT' E' \sqsubseteq (E) T' E' \sqsubseteq \\
 (TE') T' E' &\sqsubseteq (FT' E') T' E' \sqsubseteq (id T' E') T' E' \sqsubseteq \\
 (id E') T' E' &\sqsubseteq (id) T' E' \sqsubseteq (id) * FT' E' \sqsubseteq \\
 (id) * id T' E' &\sqsubseteq (id) * id E' \sqsubseteq (id) * id + TE' \sqsubseteq \\
 (id) * id + FT' E' &\sqsubseteq (id) * id + id T' E' \sqsubseteq (id) * id^* + id\$ \\
 E' &\sqsubseteq \varepsilon
 \end{aligned}$$

FIRST&FOLLOW与预测分析

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' \Rightarrow (id) * id + id$$

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

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

M - Table

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. $E' \Rightarrow \epsilon$ and $)$ in $\text{Follow}(E')$
5. Since $\$$ in $\text{Follow}(T')$, $T' \Rightarrow \epsilon$
6. Since $\$$ in $\text{Follow}(E')$, $E' \Rightarrow \epsilon$



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 α 替换A

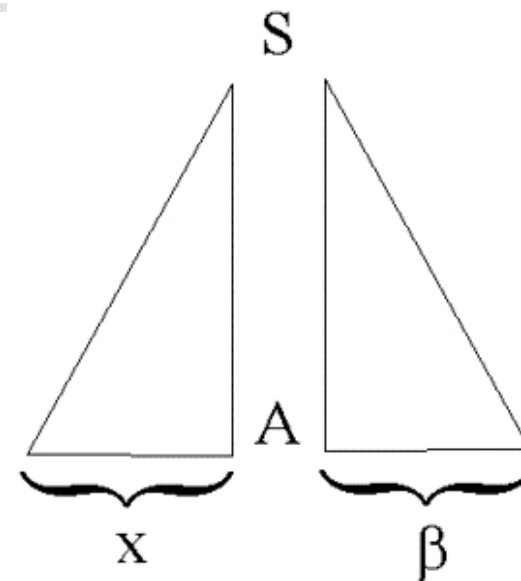
FIRST作用（续）

栈 输入缓冲区
\$βA ay\$
句型: xAβ 输入: xay

期待: $A\beta \square \alpha y \square ay$ *

即: 想把A变成a...

必然需要α的FIRST集合包含a





FOLLOW的作用

○ FOLLOW

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

FOLLOW的作用（续）

栈 输入缓冲区

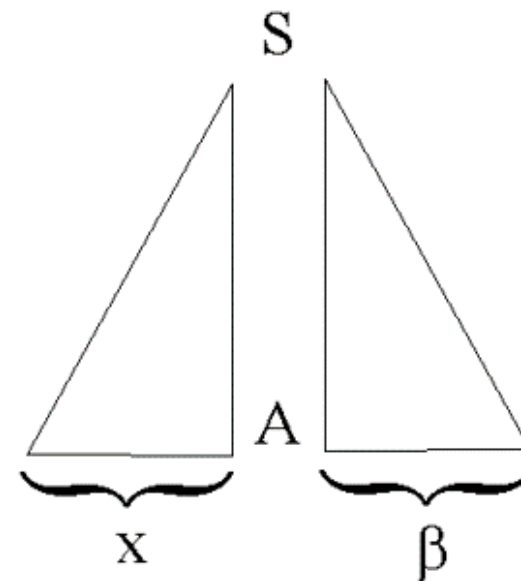
$\$ \beta A$ $by \$$

句型: $x A \beta$ 输入: $x by$

A扩展为 ϵ , 即期待: $\beta \square by$ *

即: 想把A消去, 用 β 变成 $b \dots$

存在句型: $x A by$ ——FOLLOW





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: $E \sqsubseteq TE' : First(TE') = First(T) = \{ (, id \}$

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

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

规则2

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

(规则 3) $E' \sqsubseteq e : e \in First(e) \quad T' \sqsubseteq e : e \in First(e)$

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

$M[E', \$] : E' \sqsubseteq \epsilon$ (3.2) $M[T',)] : T' \sqsubseteq \epsilon$ (3.1)

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

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

例4.19

$S \square iEtSS'$ a $S' \square eS \mid \varepsilon$ $E \square b$	$First(S) = \{i, a\}$ $First(S') = \{e, \varepsilon\}$ $First(E) = \{b\}$	$Follow(S) = \{e, \$ \}$ $Follow(S') = \{e, \$ \}$ $Follow(E) = \{t\}$
---	---	--

$S \square iEtSS'$
 $First(iEtSS') = \{i\}$

$S \square a$
 $First(a) = \{a\}$

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

$S' \square eS$
 $First(eS) = \{e\}$

$S \square \varepsilon$
 $First(\varepsilon) = \{\varepsilon\}$

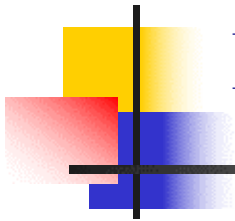
$Follow(S') = \{e, \$ \}$

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



4.4.7 LL(1)文法

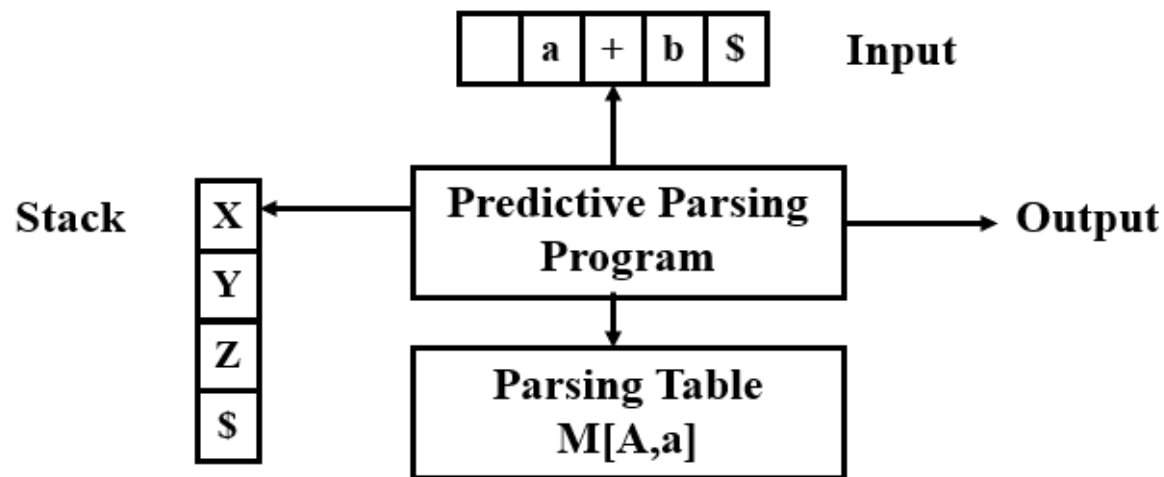
- 例4.19, $M[S', e]$ 有两个值
- 预测分析表项无多值——LL(1)文法
 - L: 由左至右扫描输入
 - L: 构造最左推导
 - 1: 向前搜索一个输入符号, 结合栈中符号,
即可确定分析器动作



LL(1)文法的特性

1. 无二义性，无左递归
2. 若 $A \Rightarrow \alpha \beta$ ，则
 1. α 、 β 推导出的符号串，不能以同样的终结符 a 开头
 2. α 、 β 至多有一个可推导出 ϵ
 3. 若 $\alpha \beta^* \epsilon$ ， $\text{FIRST}(\alpha) \cap \text{FOLLOW}(A) = \Phi$
- 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 $A \rightarrow \epsilon$, 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

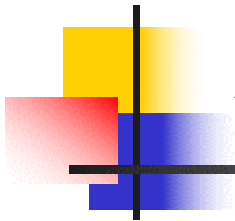
Non-terminal	INPUT SYMBOL					
	id	+	*	()	\$
E	$E \rightarrow TE'$	<u> </u>	<u> </u>	$E \rightarrow TE'$	<u>sync</u>	<u>sync</u>
E'	<u> </u>	$E' \rightarrow +TE'$	<u> </u>	<u> </u>	$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$	<u>sync</u>	<u> </u>	$T \rightarrow FT'$	<u>sync</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
SE	+ id * + id\$	error, skip +
SE	id * + id\$	id ∈ FIRST(E)
SE'T	id * + id\$	
SE'T'F	id * + id\$	
SE'T'id	id * + id\$	
SE'T'	* + id\$	
SE'T'F*	* + id\$	
SE'T'F	+ id\$	error, M[F,+] = synch
SE'T'	+ id\$	F has been popped
SE'	+ id\$	
SE'T+	+ id\$	
SE'T	id\$	
SE'T'F	id\$	
SE'T'id	id\$	
SE'T'	\$	
SE'	\$	
\$	\$	



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>$T \rightarrow FT'$</u>	<u>synch</u>		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	<u>$F \rightarrow id$</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 \square TE'

2 E' \square +TE'

3 E' $\square \square$

4 T \square FT'

5 T' \square *FT'

6 T' $\square \square$

7 F \square (E)

8 F \square 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);  
  
    while (t == ',') {  
        Tree q;  
        t = gettok();  
        q = pointer(expr1(0));  
        p = tree(RIGHT, q->type, root(value(p)), q);  
    }  
    if (tok)  
        test(tok, stop);  
    return p;  
}
```

赋值表达式

表达式序列

错误处理



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");
```

```
}
```

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

FOLLOW中的一个单词



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;
}

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

“运算”赋值

根据不同的运算符创建相应的语法树

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 ——+、—， $k+1$ ——*、/

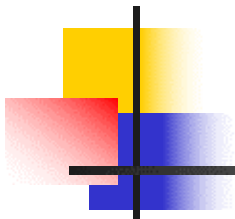
加减法表达式的结构——乘除法表

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

