第4章 语法分析

符号流 -> 语法树

- 语法分析器的类型：
  - 通用型
  - 自顶向下：通常处理LL文法
  - 自底向上：通常处理LR文法

从左到右扫描字符

L：寻找串的一个最左推导

R：反向地构造出一个最右推导序列

上下文无关文法

组成

- **终结符号**
  - 组成串的基本符号，与“词法单元名字”同义
- **非终结符号**
  - 语法变量，表示特定串的集合
  - 给出了语言的层次结构，这种层次结构是语法分析和翻译的关键
- 一个**开始符号**
  - 某个特定的非终结符号，其表示的串集合是这个文法生成的语言
- 一组**产生式**
  - 描述将终结符号和非终结符号组合成串的方法
  - 产生式左部（头）是一个非终结符号
  - 符号“→”
  - 一个由零个或多个终结符号与非终结符号组成的产生式右部（体）

句型/句子/语言：

- 句型（sentential form）：
  - 如果$S \overset{*}{\Rightarrow} \alpha$，那么α就是文法的一个句型
  - 可能既包含非终结符，又包含终结符号；可以是空串
- 句子（sentence）
  - 文法的句子就是不包含非终结符号的句型
- 语言
  - 文法G的语言就是G的句子的集合，记为L(G)
  - ω在L(G)中当且仅当ω是G的句子，即$S \overset{*}{\Rightarrow} \omega$

最左推导，最右推导，最左句型，最右句型，语法分析树

推导与语法分析树的例子

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

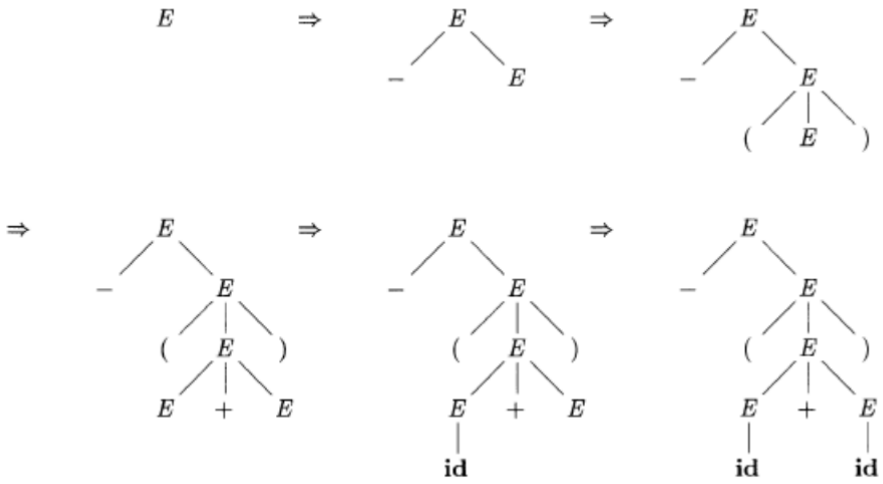


图 4-4　推导(4.8)的语法分析树序列

验证文法生成的语言

- 首先证明L(G)⊆L：G生成的每个串都在L中
- 然后证明L⊆L(G)：L的每个串都能由G生成
- 一般使用数学归纳法

不考

词法分析和语法分析比较

阶段	输入	输出	描述体系
词法分析	源程序符号串	词法单元序列	正则表达式
句法分析	词法单元序列	语法树	上下文无关文法

- 文法比正则表达式描述能力更强
- 正则表达式描述词法单元比较简洁
- 基于正则表达式构造的词法分析器效率更高
- 正则表达式适合描述词法结构，文法适合描述嵌套结构

上下文无关文法和正则表达式：

上下文无关文法比正则表达式的能力更强，即：

- 一些用文法描述的语言不能用正则文法描述
- 所有的正则语言都可以使用文法描述

不考为NFA构造等价文法

消除二义性

不考：二义性的消除方法没有规律可循，通常并不是通过改变文法来消除二义性

提取左公因子

$$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma$$

替换为

$$A \to \alpha A' \mid \gamma$$

$$A' \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

消除左递归（重点）

左递归的定义

- 左递归的定义
  - 文法中一个非终结符号A使得对某个串α，存在一个推导 $A \overset{+}{\Rightarrow} A\alpha$ 则称这个文法是左递归的
- 立即左递归
  - 如果存在$A \to A\alpha$,则称为立即左递归
- 为什么要消除左递归？
  - 自顶向下的语法分析技术不能处理左递归的文法
- 立即左递归的消除实例：

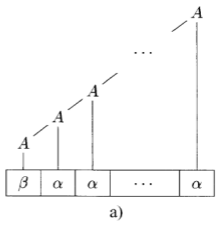
$$A \to A\alpha \mid \beta \quad \Longrightarrow \quad \begin{array}{l} A \to \beta A' \\ A' \to \alpha A' \mid \varepsilon \end{array}$$

立即左递归的消除

- 假设非终结符号A存在立即左递归的情形，假设以A为左部的规则有：

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

首先对A产生式**分组**(所有 α_i不等于 ε， β_i不以A开头)



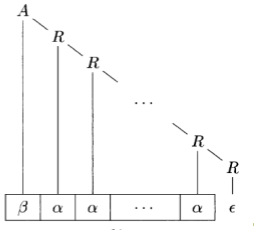
a)

可以**替换**为

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

$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon$$

- 由A生成的串总是以某个 β_i 开头，然后跟上零个或者多个 α_i 的重复



b)

消除多步左递归

- 输入：没有环$A_i \overset{+}{\Rightarrow} A_i$和 $A \to \varepsilon$
- 输出：一个等价的无左递归的文法

- 算法原理：
  - 给非终结符号**排序**，A_1，A_2，…，A_n
  - 如果只有$A_i \Rightarrow A_j$ (i<j)，则不会有左递归
  - 如果发现$A_i \Rightarrow A_j$ (i>j)，**代入** A_j 的当前产生式，若替换后有 A_i 的直接左递归，再**消除**

伪代码：

- 输入：没有环和ε产生式的文法G
- 输出：等价的无左递归的文法
- 步骤：

将文法的非终结符号任意排序为$A_1,A_2,\ldots,A_n$

```
for i=1 to n do {
    for j = 1 to i-1 do
    {
    将形如Ai→ Ajγ的产生式替换为Ai→δ1γ| δ2γ | … |δkγ，
    其中Aj→ δ1| δ2|…| δk是以Aj为左部的所有产生式；
    }
    消除Ai的立即左递归；
}
```

示例

- S → Aa | b,　A → Ac | Sd | ε
- 排序：　S　A
- 替换：
  - i=1，没有处理
  - i=2, 替换 A→Sd 中的 S，得到 A → Ac | Aad | bd | ε
- 消除立即左递归

$$S \rightarrow A\ a \mid b$$
$$A \rightarrow b\ d\ A' \mid A'$$
$$A' \rightarrow c\ A' \mid a\ d\ A' \mid \epsilon$$

自顶向下语法分析

在推导的每一步，对非终结符号A，应用哪个产生式，以可能产生于输入串相匹配的终结符号串

预测分析技术

通过在输入中向前看固定多个符号来选择正确的产生式

通常情况下，我们只需要向前看一个符号

递归下降分析一般只适合于每个子表达式的第一个终结符能够为产生式选择提供足够信息的那些文法

给出两个与文法相关的两个函数 □ FIRST □ FOLLOW

基于上述两个函数，可以根据下一个输入符号来 选择应用哪个产生式

First函数计算

First定义：

- 定义：可从α推导得到的串的首符号的集合，其中α是任意的文法符号串。如果$\alpha \stackrel{*}{\Rightarrow} \varepsilon$，那么ε也在FIRST(α)中
- FIRST函数的意义
  - 如果两个A产生式 $A \rightarrow \alpha \mid \beta$，其中First($\alpha$)和First($\beta$)是不相交的集合。下一个输入符号是a，若a∈ First(α)，则选择$A \rightarrow \alpha$，若a∈ First(β)，则选择$A \rightarrow \beta$
  - *用于预测产生式的选择*

First计算：

- 对于文法符号X的FIRST(X)，通过不断应用下列规则，直到没有新的终结符号或者ε可以加入到任何FIRST集合为止
  - 如果X是终结符号，那么FIRST(X)={X}
  - 如果X是非终结符号，且有规则X → a...,那么将a添加到FIRST(X)中; 如果X → ε，那么ε也在FIRST(X)中
  - 对于规则X → $Y_1Y_2...Y_n$, 把FIRST(Y_1)中的非ε符号添加到FIRST(X)中。如果ε在FIRST(Y_1)中，把FIRST(Y_2)中的非ε符号添加到FIRST(X)中...；如果ε在FIRST(Y_n)中，把ε添加到FIRST(X)中

示例：

$E \rightarrow T\ E'$

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

$T \rightarrow F\ T'$

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

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

- First(F)=First(T)=First(E)={(,id}
- First(E')={+, ε}
- First(T')={*, ε}
- First(TE')=First(T)={(,id}
- First(+TE')={+}
- ……

Follow函数计算

Follow定义：

- 对于非终结符号A，FOLLOW(A)定义为可能在某些句型中紧跟在A右边的终结符号的集合
  - 例如S $\stackrel{*}{\Rightarrow}$αAaβ，终结符号a∈Follow(A)
- 如果A是某些句型的最右符号，那么$ ∈Follow(A)。 $是特殊的输入串“结束标记”
- FOLLOW函数的意义：
  - 如果A→α，当α→ε或α $\stackrel{*}{\Rightarrow}$ε时，FOLLOW(A)可以帮助我们做出选择恰当的产生式
  - 例如：如果A→ α, b属于FOLLOW(A)，如果α $\stackrel{*}{\Rightarrow}$ε，则若当前输入符号是b，可以选择A → α，因为A最终到达了ε，而且后面跟着b

Follow计算：

- 计算各个非终结符号*A*的FOLLOW(*A*)集合，不断应用下列规则，直到没有新的终结符号可以被加入到任意FOLLOW集合中
  - 将$放入FOLLOW(*S*)，*S*是开始符号，而$是输入串的结束标记
  - 如果存在产生式*A* → *αBβ*，那么First(*β*)中除ε之外的所有符号都在Follow(B)中
  - 如果存在一个产生式*A* → *αB*，或存在产生式*A* → *αBβ*且First(*β*)包含ε，那么Follow(*A*)中的所有符号都在Follow(*B*)中

示例：

- 文法

  E → T E'　　E' → +T E' | ε　　T → F T'

  T' → *F T' | ε　　F → (E) | i

- FOLLOW(E)={) ， $}　← FIRST( )) U {$}
- FOLLOW(E')={) ， $}　← FOLLOW(E)
- FOLLOW(T)={+，)， $}　← FIRST(E') U FOLLOW(E')
- FOLLOW(T')={+，)， $}　← FOLLOW(T)
- FOLLOW(F)={*， +，)， $)　← FIRST(T') U FOLLOW(T)

  - 将$放入FOLLOW(*S*)，*S*是开始符号，而$是输入串的结束标记
  - 如果存在产生式*A* → *αBβ*，那么First(*β*)中除ε之外的所有符号都在Follow(B)中
  - 如果存在一个产生式*A* → *αB*，或存在产生式*A* → *αBβ*且First(*β*)包含ε，那么Follow(*A*)中的所有符号都在Follow(*B*)中

预测分析表

- 输入：文法G
- 输出：预测分析表M

- 方法：对于文法G的每个产生式A→α，进行如下处理
  - 对于First(α)中的每个终结符号a，将A→α加入到M[A,a]
  - 如果ε在First(α)中，那么对于Follow(A)中的每个终结符号b，将A→α加入到M[A,b]中
  - 如果ε在First(α)中，且$在Follow(A)中，将A→α加入到M[A,$]中

- 完成上述操作后，若M[A,a]中没有产生式，填为Error

分析表驱动预测分析

已匹配	栈	输入	动作
	$E\$$	$\mathbf{id}+\mathbf{id}*\mathbf{id}\$$	
	$TE'\$$	$\mathbf{id}+\mathbf{id}*\mathbf{id}\$$	输出 $E \to TE'$
	$FT'E'\$$	$\mathbf{id}+\mathbf{id}*\mathbf{id}\$$	输出 $T \to FT'$
	$\mathbf{id}\ T'E'\$$	$\mathbf{id}+\mathbf{id}*\mathbf{id}\$$	输出 $F \to \mathbf{id}$
$\mathbf{id}$	$T'E'\$$	$+\,\mathbf{id}*\mathbf{id}\$$	匹配 $\mathbf{id}$
$\mathbf{id}$	$E'\$$	$+\,\mathbf{id}*\mathbf{id}\$$	输出 $T' \to \epsilon$
$\mathbf{id}$	$+\,TE'\$$	$+\,\mathbf{id}*\mathbf{id}\$$	输出 $E' \to +TE'$
$\mathbf{id}+$	$TE'\$$	$\mathbf{id}*\mathbf{id}\$$	匹配 $+$

自底向上语法分析

从叶子（输入串中的终结符号，将位于分析树的底端）开始，向上到达根结点

在每一步的归约中，一个与某产生式体相匹配的特定子串被替换为该产生式头部的非终结符号，一次归约实质上是一个推导的反向操作。

句柄

句柄

重要

- 最右句型γ的一个句柄
  - 满足下述条件的产生式A→β及串β在γ中出现的位置
  - 条件：将这个位置上的β替换为A之后得到的串是γ的某个最右推导序列中出现在位于γ之前的最右句型
- 通俗地说
  - 句柄是最右推导的反向过程中被规约的那些部分
- 句柄的作用
  - 对句柄的规约，代表了相应的最右推导中的一个反向步骤

移进归约语法分析框架

- 移入：将下一个输入符号移动到栈顶
- 归约：将句柄归约为相应的非终结符号
  - **句柄总是在栈顶**
  - 具体操作时弹出句柄，压入被归约到的非终结符号
- 接受：宣布分析过程成功完成
- 报错：发现语法错误，调用错误恢复子程序

例子:

栈	输入	动作
$	$\mathbf{id}_1 * \mathbf{id}_2$ $	移入
$ $\mathbf{id}_1$	$* \mathbf{id}_2$ $	按照 $F \to \mathbf{id}$ 归约
$ F	$* \mathbf{id}_2$ $	按照 $T \to F$ 归约
$ T	$* \mathbf{id}_2$ $	移入
$ $T *$	$\mathbf{id}_2$ $	移入
$ $T * \mathbf{id}_2$	$	按照 $F \to \mathbf{id}$ 归约
$ $T * F$	$	按照 $T \to T * F$ 归约
$ T	$	按照 $E \to T$ 归约
$ E	$	accept

LR语法分析技术

- LR(k)的语法分析概念
  - L表示最左扫描，R表示反向构造出最右推导
  - k表示最多向前看k个符号
- 当k的数量增大时，相应的语法分析器的规模急剧增大
  - K=2时，程序设计语言的语法分析器的规模通常非常庞大
  - 当k=0、1时已经可以解决很多语法分析问题，因此具有实践意义
  - 因此，我们只考虑k<=1的情况

LR(0)项

LR(0)项

- 文法的一个产生式加上在其产生式体中某处的一个点
  - **A →.XYZ，A → X.YZ，A → XY.Z，A → XYZ.**
  - 注意：**A → ε**只对应一个项**A →.**
- 直观含义
  - 项**A → α.β**表示已经扫描/归约到了**α**，并期望接下来的输入中经过扫描/归约得到**β**，然后把**αβ**归约到**A**
  - 如果**β**为空，表示我们可以把**α**归约为**A**
- 项也可以用一对整数表示
  - **(i,j)**表示第**i**条规则，点位于右部第**j**个位置

增广文法

- 增广文法
  - **G的增广文法G'是在G中增加新开始符号S'，并加入产生式S'→S而得到的**
  - **G'和G接受相同的语言，且按照S'→ S进行归约实际上就表示已经将输入符号串归约成为开始符号**
  - **引入的目的是告诉语法分析器何时宣布接受输入符号串，即用S'→S进行归约时，表明分析结束。**

即用S'→S进行归约时，表明分析结束

CLOSURE和GOTO

- **如果I是文法G的一个项集，那么CLOSURE(I)就是根据下列规则从I构造得到的项集**
  - **将I中的各个项加入到CLOSURE(I)中**
  - **如果A → α.Bβ在CLOSURE(I)中，那么对B的任意产生式B → γ，将B →.γ加到CLOSURE(I)中**
  - **不断重复第二步，直到收敛**
- **第二步的意义**
  - **项A → α.Bβ表示期望在接下来的输入中归约到B**
  - **显然，要归约到B，首先要扫描归约到B的某个产生式的右部**
  - **因此对每个产生式B → γ，加入B →.γ**
    - **表示它期望能够扫描归约到γ**

GOTO：

- I是一个项集，X是一个文法符号，GOTO(I,X)定义为I中所有形如的项[A → α·Xβ]所对应的项[A → αX·β]的集合的闭包
  - 根据项的历史-期望的含义，GOTO(I,X)表示读取输入中的X或者归约到一个X之后的情况
  - GOTO(I,X)定义了LR(0)自动机中状态I在X之上的转换
- 例如：
  - I={[E' → E.], [E → E.+T]}
  - GOTO(I,+)计算如下
    - I中只有一个项的点后面跟着+，对应的项为[E → E+.T]
    - CLOSURE({[E → E+.T]}) = {[E → E+.T]，[T →.T*F], [T→.F]，[F →.(E)]，[F →.id]}

LR(0)自动机

- 构造方法
  - 规范LR(0)项集族中的项集可以作为LR(0)自动机的状态
  - GOTO(I,X)=J，则从I到J有一个标号为X的转换
  - 初始状态为CLOSURE({S' →.S})对应的项集
  - 接受状态：包含形如A → α.的项集对应的状态

作用：

- 假设文法符号串γ使LR(0)自动机从开始状态运行到状态（项集）j
  - 如果j中有一个形如A → α.的项，那么
    - 在γ之后添加一些终结符号可以得到一个最右句型
    - α是γ的后缀，且A → α是这个句型的句柄
    - 表示可能找到了当前最右句型的句柄
  - 如果j中存在一个项B → α.Xβ，那么
    - 在γ之后添加Xβ，然后再添加一个终结符号串可得到一个最右句型
    - 在这个句型中B → αXβ是句柄
    - 此时表示还没有找到句柄，需要移入

LR语法分析表

- 两个部分：动作ACTION，转换GOTO
- ACTION有两个参数：状态i、终结符号a
  - 移入j：j是一个状态。把j压入栈
  - 归约A → β：把栈顶的β归约为A
  - 接受：接受输入、完成分析
  - 报错：在输入中发现语法错误
- 状态集上的GOTO函数
  - 如果GOTO[I_i,A]=I_j，那么GOTO[i,A] = j

例子:

LR分析

- 输入：**id*id+id**

状态	ACTION						GOTO		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

	栈	符号	输入	动作
(1)	0		id * id + id $	移入
(2)	0 5	id	* id + id $	根据 $F \to$ id 归约
(3)	0 3	F	* id + id $	根据 $T \to F$ 归约
(4)	0 2	T	* id + id $	移入
(5)	0 2 7	T *	id + id $	移入
(6)	0 2 7 5	T * id	+ id $	根据$F \to$ id 归约
(7)	0 2 7 10	T * F	+ id $	根据$T \to T * F$归约
(8)	0 2	T	+ id $	根据$E \to T$ 归约
(9)	0 1	E	+ id $	移入
(10)	0 1 6	E +	id $	移入
(11)	0 1 6 5	E + id	$	根据 $F \to$ id 归约
(12)	0 1 6 3	$E + F$	$	根据 $T \to F$ 归约
(13)	0 1 6 9	$E + T$	$	根据 $E \to E + T$归约
(14)	0 1	E	$	接受