



山东大学  
SHANDONG UNIVERSITY

## 编译原理

# 第六章 属性文法和语法制导翻译

授 课 教 师 : 余仲星  
手 机 : 15866821709 (微信同号)  
邮 箱 : zhongxing.yu@sdu.edu.cn

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

### □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

### □ 6.3 S-属性文法的自下而上计算

### □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

### □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

## 6.1 属性文法

- **属性文法**在1968年由Knuth提出，也称**属性翻译文法**，是在**上下文无关文法**的基础上，为每个文法符号 ( $V_T \cup V_N$ ) 配备**若干属性**。
  - **属性代表与文法符号相关的信息**，如其类型、值、代码序列、符号表内容等；
  - 属性与变量一样，**可以进行计算和传递**；
  - 属性的**加工过程即语义的处理过程**；
  - 对文法的每个产生式都配备了一组属性的**计算规则**，称为**语义规则**。

## 6.1 属性文法

□ 每个产生式 $A \rightarrow \alpha$ 都有一套与之关联的语义规则，规则形式为

$$b = f(c_1, c_2, \dots, c_k)$$

这里 $f$ 是一个函数，满足以下之一：

- $b$ 是 $A$ 的一个综合属性，且 $c_1, c_2, \dots, c_k$ 是产生式右边文法符号的属性；
- $b$ 是产生式右边某个文法符号的一个继承属性，且 $c_1, c_2, \dots, c_k$ 是 $A$ 或产生式右部任何文法符号的属性。

□ 以上两种情况，都称为属性 $b$ 依赖于属性 $c_1, c_2, \dots, c_k$ 。

□ 注意：

- 终结符只有综合属性，它们由词法分析器提供；
- 非终结符既可以有综合属性也可以有继承属性，文法开始符号的所有继承属性作为属性计算前的初始值。

## 6.1 属性文法

- 出现在产生式**右边的继承属性**和出现在产生式**左边的综合属性**都必须提供一个计算规则
    - 属性计算规则中，**只能使用相应产生式中的文法符号的属性**，这有助于在产生式范围内“封装”属性的依赖性。
  - 出现在产生式**左边的继承属性**和出现在产生式**右边的综合属性**不由所给产生式的属性规则进行计算
    - 它们由**其它产生式**的属性规则计算或者由属性计算器的**参数**提供。
- 【例6.1】**  $A, B, C \in V_N$ ,  $A$ 有继承属性 $a$ 和综合属性 $b$ ,  $B$ 有综合属性 $c$ ,  $C$ 有继承属性 $d$ , 产生式 $A \rightarrow BC$ 可能有规则:

$$C.d = B.c + 1$$

$$A.b = A.a + B.c$$

- $A.a$ 和 $B.c$ 在其它地方计算。

## 6.1 属性文法

【例6.2】一个简单台式计算器的属性文法：

$L \rightarrow E$        $print(E.val)$  // 只对应一个动作，可以认为定义了 $L$ 的一个虚属性

$E \rightarrow E_1 + T$        $E.val = E_1.val + T.val$  // 一个产生式出现了两个 $E$ ，用下标区分

$E \rightarrow T$        $E.val = T.val$

$T \rightarrow T_1 * F$        $T.val = T_1.val * F.val$

$T \rightarrow F$        $T.val = F.val$

$F \rightarrow (E)$        $F.val = E.val$

$F \rightarrow digit$        $F.val = digit.lexval$  //  $digit.lexval$ 由词法分析器提供

# 综合属性

- 一个结点的综合属性值由其子结点的属性值确定;
- 通常使用自底向上的方法计算综合属性值;
- 仅仅使用综合属性的文法称为S-属性文法。

## 【例6.3】 $3*5+4$ 属性值计算

$L \rightarrow E$        $print(E.val)$

$E \rightarrow E_1 + T$        $E.val = E_1.val + T.val$

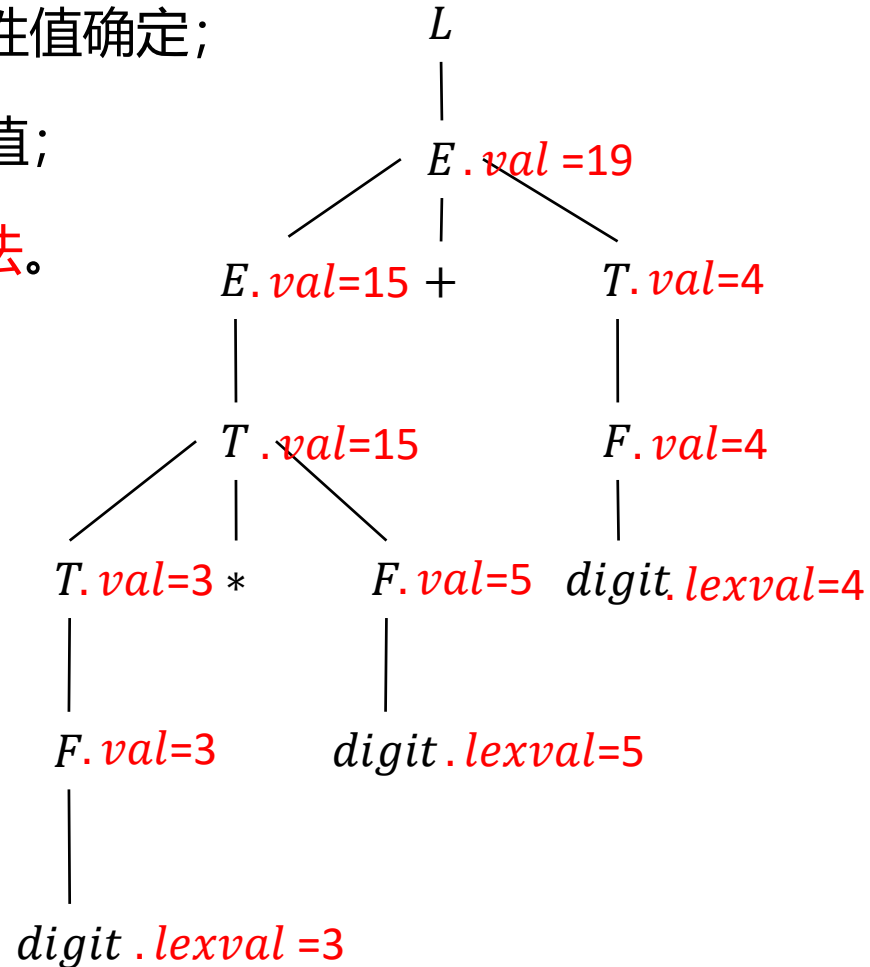
$E \rightarrow T$        $E.val = T.val$

$T \rightarrow T_1 * F$        $T.val = T_1.val * F.val$

$T \rightarrow F$        $T.val = F.val$

$F \rightarrow (E)$        $F.val = E.val$

$F \rightarrow digit$        $F.val = digit.lexval$





# 继承属性

□ 一个结点的继承属性值由其父和/或兄结点的属性值确定。

【例6.4】数据类型跟踪:  $real\ id_1, id_2, id_3$

$D \rightarrow TL$        $L.type = T.type$

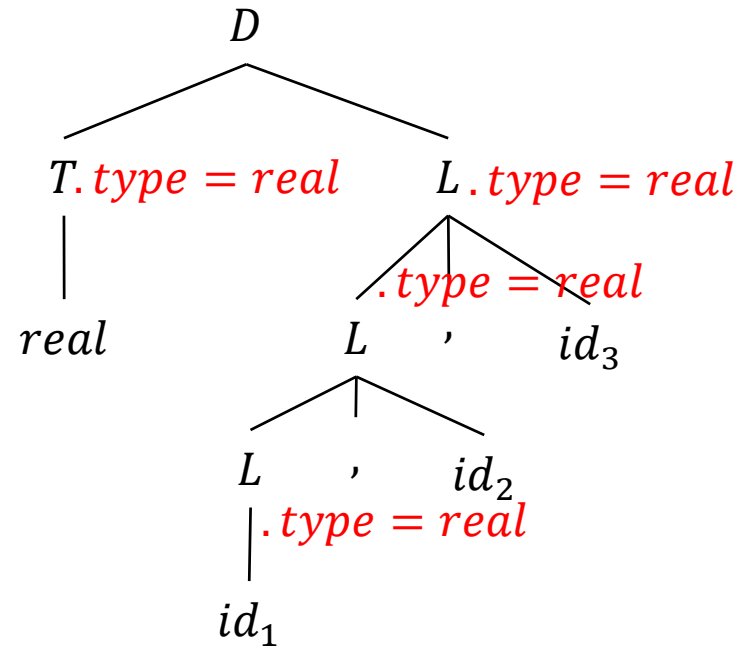
$T \rightarrow int$        $T.type = integer$

$T \rightarrow real$        $T.type = real$

$L \rightarrow L_1, id$        $L_1.type = T.type$

$addType(id.entry, L.type)$

$L \rightarrow id$        $addType(id.entry, L.type)$



# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

## 6.2 基于属性文法的处理方法

- 由源程序的语法结构所驱动的处理方法，称为**语法制导翻译法**，其语义规则计算可能：
  - 产生代码；
  - 在符号表中存放信息；
  - 给出错误信息；
  - 执行任何其它动作。

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

### ➤ 6.2.1 依赖图

### ➤ 6.2.2 树遍历的属性计算方法

### ➤ 6.2.3 一遍扫描的处理方法

### ➤ 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

### ➤ 6.4.1 翻译模式

### ➤ 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

### ➤ 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作

### ➤ 6.5.2 分析栈中的继承属性

### ➤ 6.5.3 模拟继承属性的计算

### ➤ 6.5.4 用综合属性代替继承属性

## 6.2.1 依赖图

- 如果在一颗语法树中，一个结点的属性 $b$ 依赖于属性 $c$ ，那么这个结点处计算 $b$ 的语义规则必须在确定 $c$ 的语义规则之后使用。
- 依赖图：是一个表示语法树中结点间相互依赖关系的有向图。
  - $b = f(c_1, c_2, \dots, c_k)$ ，则属性 $b$ 依赖于属性 $c_i$ ，从 $c_i$ 向 $b$ 画有向边；
  - 过程调用的语法规则则生成一个虚综合属性 $b = f(c_1, c_2, \dots, c_k)$ 。

for 语法树的每个结点 $n$

for 结点 $n$ 的文法符号的每个属性 $a$

为 $a$ 在依赖图中建立一个结点；

for 语法树的每个结点 $n$

for 结点 $n$ 所用产生式对应的每个语义规则 $b = f(c_1, c_2, \dots, c_k)$

for  $i = 1 \dots k$

从 $c_i$ 向 $b$ 构造一条有向边；

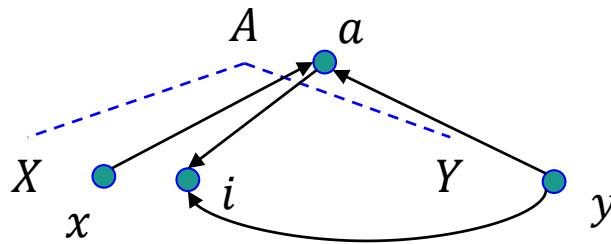
## 6.2.1 依赖图

### 【例6.5】

$$A \rightarrow XY$$

$$A.a = f(X.x, Y.y)$$

$$X.i = g(A.a, Y.y)$$



## 【例6.6】 $3*5+4$ 属性值计算

$L \rightarrow E$        $print(E.val)$

$E \rightarrow E_1 + T$        $E.val = E_1.val + T.val$

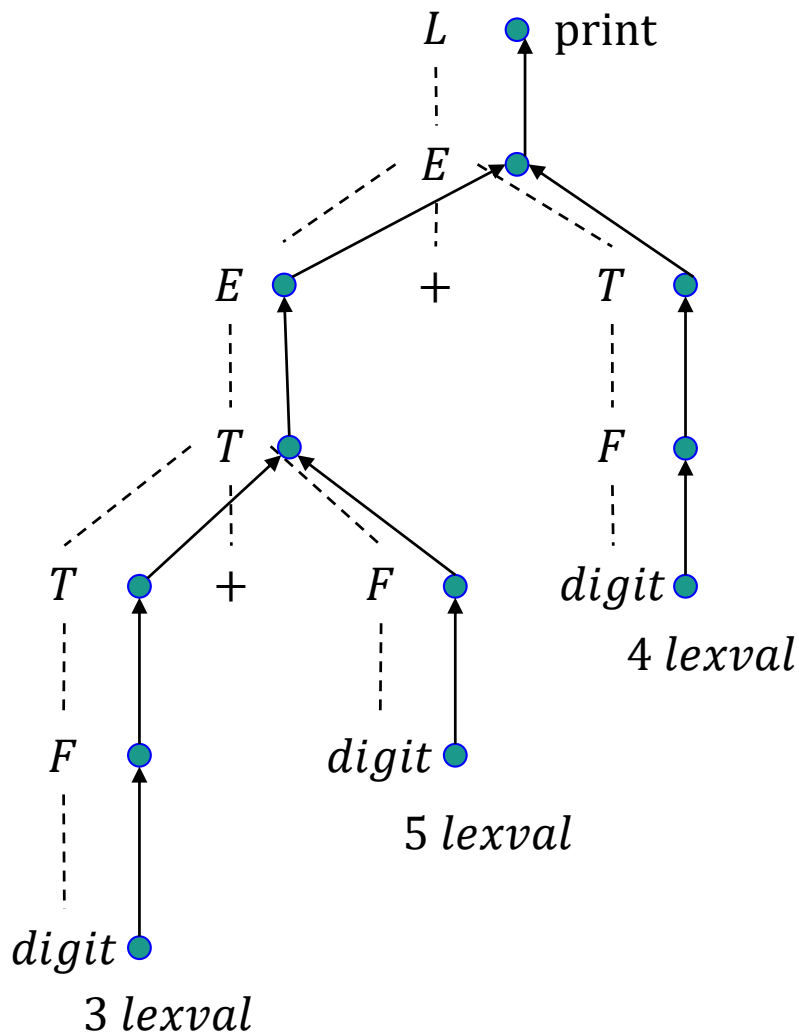
$E \rightarrow T$        $E.val = T.val$

$T \rightarrow T_1 * F$        $T.val = T_1.val * F.val$

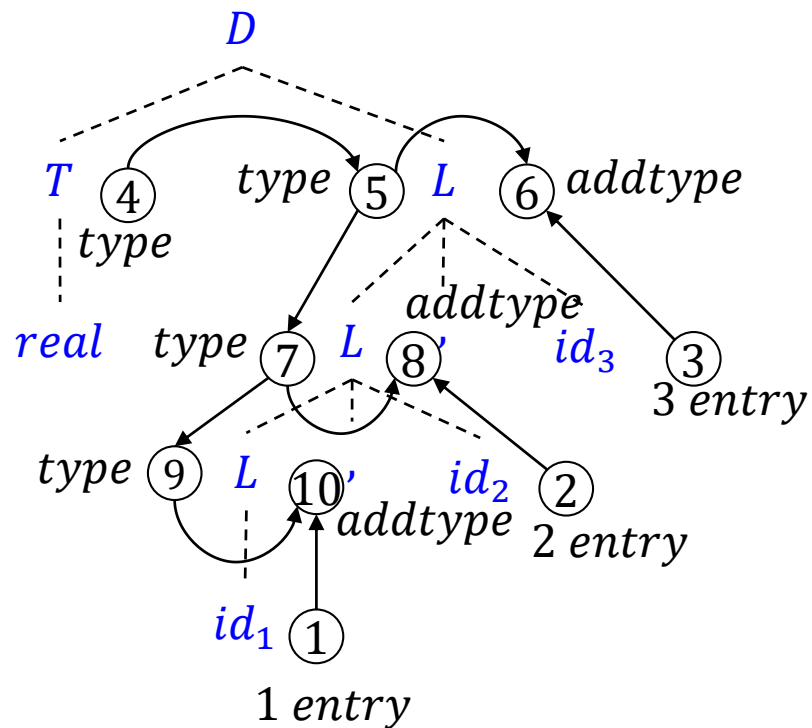
$T \rightarrow F$        $T.val = F.val$

$F \rightarrow (E)$        $F.val = E.val$

$F \rightarrow digit$        $F.val = digit.lexval$



**【例6.7】数据类型跟踪：**  $real\ id_1, id_2, id_3$

$$D \rightarrow TL \quad L.type = T.type$$
$$T \rightarrow int \qquad T.type = integer$$
$$T \rightarrow real \quad T.type = real$$
$$L \rightarrow L_1, id \quad L_1.type = L.type$$
$$addType(id.entry, L.type)$$
$$L \rightarrow id \quad addType(id.entry, L.type)$$




## 6.2.1 依赖图

- 如果一个属性文法不存在属性之间的循环依赖关系，称该文法为良定义的
  - 为了设计编译程序，我们只处理良定义的属性文法。
- 属性的计算次序
  - 一个有向非循环图的拓扑序是图中结点的任何顺序 $m_1, m_2, \dots, m_k$ ，使得边必须是从序列种前面的结点指向后面的结点；
  - 即：如果 $m_i \rightarrow m_j$ 是 $m_i$ 到 $m_j$ 的一条边，则序列中 $m_i$ 必须在 $m_j$ 之前；

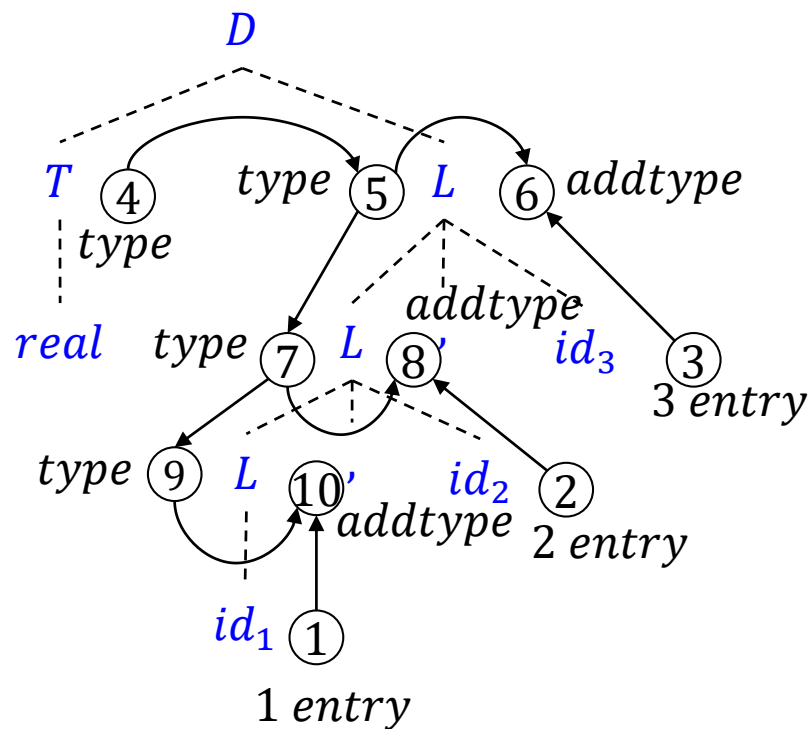
**【例6.8】数据类型跟踪：**  $real\ id_1, id_2, id_3$

■ 属性的计算次序：结点1-10，用 $a_i$ 表示结

点 $i$ 的相关属性, 得到如下程序:

$$a_4 = real;$$
$$a_5 = a_4;$$

```
addType(id3.entry, a5);
```

$$a_7 = a_5;$$
$$addType(id_2.entry, a_7);$$
$$a_9 = a_7;$$
$$addType(id_1.entry, a_9);$$


# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

### ➤ 6.2.1 依赖图

### ➤ 6.2.2 树遍历的属性计算方法

### ➤ 6.2.3 一遍扫描的处理方法

### ➤ 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

### ➤ 6.4.1 翻译模式

### ➤ 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

### ➤ 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作

### ➤ 6.5.2 分析栈中的继承属性

### ➤ 6.5.3 模拟继承属性的计算

### ➤ 6.5.4 用综合属性代替继承属性

## 6.2.2 树遍历的属性计算方法

□ 无循环属性文法计算，最坏情况时间复杂度 $O(n^2)$

while 还有未被计算的属性

VisitNode(S); // S是开始符号

void VisitNode(N) {

if ( $N \in V_N$ ) { // 设其产生式为 $N \rightarrow X_1X_2 \dots X_m$

for  $i = 1 \dots m$  {

if ( $X_i \in V_N$ ) {

计算 $X_i$ 所有能计算的继承属性;

VisitNode( $X_i$ )

} /\* end of if \*/ } /\* end of for \*/

计算 $N$ 所有能够计算的综合属性;

} /\* end of if \*/ } /\* end of VisitNode \*/

【例6.9】输入串 $xyz$ , 初始值 $S.a = 0$

$S \rightarrow XYZ$        $Z.h = S.a$   
 $X.c = Z.g$   
 $S.b = X.d - 2$   
 $Y.e = S.b$

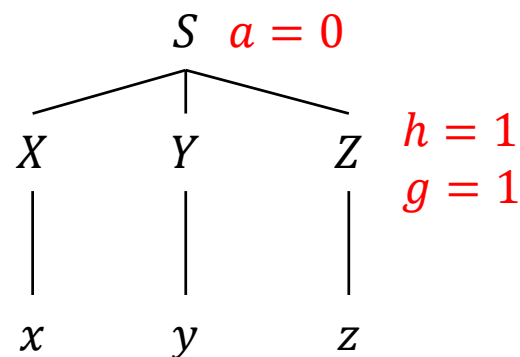
$X \rightarrow x$        $X.d = 2 * X.c$

$Y \rightarrow y$        $Y.f = Y.e * 3$

$Z \rightarrow z$        $Z.g = Z.h + 1$

其中:

- $S$ 有继承属性 $a$ , 综合属性 $b$
- $X$ 有继承属性 $c$ , 综合属性 $d$
- $Y$ 有继承属性 $e$ , 综合属性 $f$
- $Z$ 有继承属性 $h$ , 综合属性 $g$



第1次遍历:

VisitNode( $S$ )

$X.c$ 不能计算

VisitNode( $X$ )     $X.d$ 不能计算

$Y.e$ 不能计算

VisitNode( $Y$ )     $Y.f$ 不能计算

$Z.h = 0$

VisitNode( $Z$ )     $Z.g = 1$

$S.b$ 不能计算

【例6.9】输入串 $xyz$ , 初始值 $S.a = 0$

$S \rightarrow XYZ$        $Z.h = S.a$   
 $X.c = Z.g$   
 $S.b = X.d - 2$   
 $Y.e = S.b$

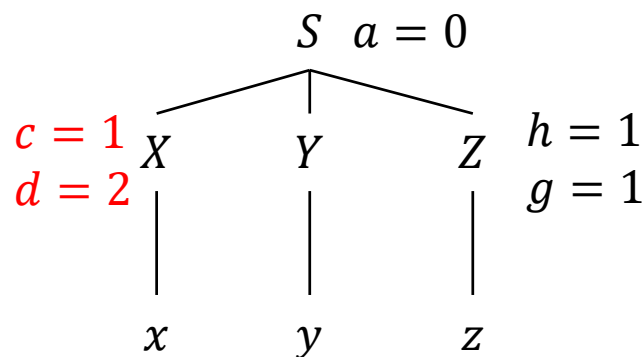
$X \rightarrow x$        $X.d = 2 * X.c$

$Y \rightarrow y$        $Y.f = Y.e * 3$

$Z \rightarrow z$        $Z.g = Z.h + 1$

其中:

- $S$ 有继承属性 $a$ , 综合属性 $b$
- $X$ 有继承属性 $c$ , 综合属性 $d$
- $Y$ 有继承属性 $e$ , 综合属性 $f$
- $Z$ 有继承属性 $h$ , 综合属性 $g$



第2次遍历:

VisitNode( $S$ )

$X.c = 1$

VisitNode( $X$ )     $X.d = 2$

$Y.e$ 不能计算

VisitNode( $Y$ )     $Y.f$ 不能计算

$Z.h = 0$

VisitNode( $Z$ )     $Z.g = 1$

$S.b$ 不能计算

【例6.9】输入串 $xyz$ , 初始值 $S.a = 0$

$S \rightarrow XYZ$        $Z.h = S.a$   
 $X.c = Z.g$   
 $S.b = X.d - 2$   
 $Y.e = S.b$

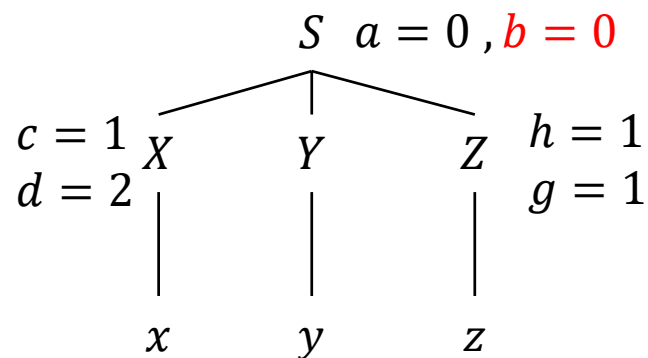
$X \rightarrow x$        $X.d = 2 * X.c$

$Y \rightarrow y$        $Y.f = Y.e * 3$

$Z \rightarrow z$        $Z.g = Z.h + 1$

其中:

- $S$ 有继承属性 $a$ , 综合属性 $b$
- $X$ 有继承属性 $c$ , 综合属性 $d$
- $Y$ 有继承属性 $e$ , 综合属性 $f$
- $Z$ 有继承属性 $h$ , 综合属性 $g$



第3次遍历:

VisitNode( $S$ )

$X.c = 1$

VisitNode( $X$ )     $X.d = 2$

$Y.e$ 不能计算

VisitNode( $Y$ )     $Y.f$ 不能计算

$Z.h = 0$

VisitNode( $Z$ )     $Z.g = 1$

$S.b = 0$

【例6.9】输入串 $xyz$ , 初始值 $S.a = 0$

$S \rightarrow XYZ$        $Z.h = S.a$   
 $X.c = Z.g$   
 $S.b = X.d - 2$   
 $Y.e = S.b$

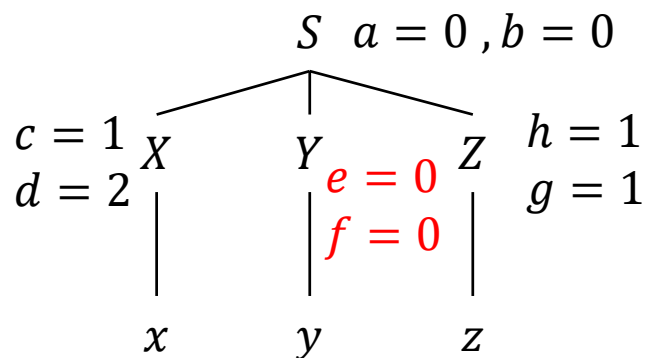
$X \rightarrow x$        $X.d = 2 * X.c$

$Y \rightarrow y$        $Y.f = Y.e * 3$

$Z \rightarrow z$        $Z.g = Z.h + 1$

其中:

- $S$ 有继承属性 $a$ , 综合属性 $b$
- $X$ 有继承属性 $c$ , 综合属性 $d$
- $Y$ 有继承属性 $e$ , 综合属性 $f$
- $Z$ 有继承属性 $h$ , 综合属性 $g$



第4次遍历:

VisitNode( $S$ )

$X.c = 1$

VisitNode( $X$ )     $X.d = 2$

$Y.e = 0$

VisitNode( $Y$ )     $Y.f = 0$

$Z.h = 0$

VisitNode( $Z$ )     $Z.g = 1$

$S.b = 0$



# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

## 6.2.3 一遍扫描的处理方法

### □ 一遍扫描与树遍历的不同之处

- 在语法分析的同时计算属性值，而不是语法分析构造语法树之后进行属性计算，而且无需构造实际的语法树（如需要也可以构建）；
- 当一个属性值不再用于计算其它属性值时，编译程序不必再保留这个属性值（如果需要也可以保留）。

### □ 一遍扫描的影响因素

- 所采用的语法分析方法；
- 属性的计算顺序。

### □ 一遍扫描的情况

- S-属性文法适合于一遍扫描的自下而上分析；
- L-属性文法适合于一遍扫描的自上而下分析。

## 6.2.3 一遍扫描的处理方法

- **语法制导翻译**：为文法的每个产生式配上一组语义规则，并且在语法分析的同时执行这些语义规则，完成有关语义分析和代码生成的工作。
  - 在自上而下的分析中，当一个产生式匹配输入串成功时执行；
  - 在自下而上的分析中，当一个产生式被用于归约时执行。

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

### ➤ 6.2.1 依赖图

### ➤ 6.2.2 树遍历的属性计算方法

### ➤ 6.2.3 一遍扫描的处理方法

### ➤ 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

### ➤ 6.4.1 翻译模式

### ➤ 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

### ➤ 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作

### ➤ 6.5.2 分析栈中的继承属性

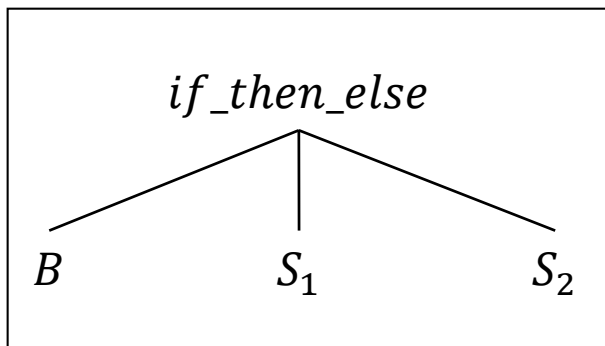
### ➤ 6.5.3 模拟继承属性的计算

### ➤ 6.5.4 用综合属性代替继承属性

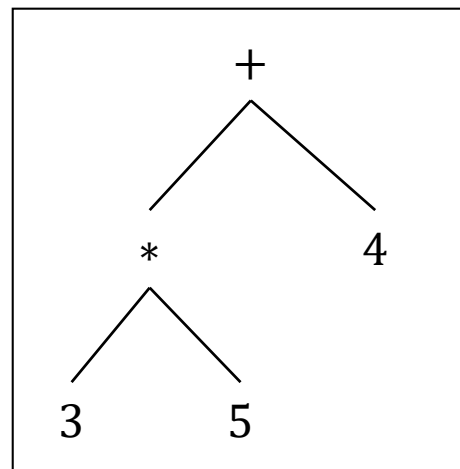
## 6.2.4 抽象语法树

❑ **抽象语法树 (Abstract Syntax Tree)**：在语法树中去掉那些对翻译不必要的信息，从而获得更有效的源程序中间表示。

- 操作符和关键字都不作为叶结点出现，而是作为内部结点；
- 语法制导翻译既可以基于语法分析树，也可以基于抽象语法树进行。



$S \rightarrow \text{if } B \text{ then } S_1 \text{ else } S_2$



$3 * 5 + 4$

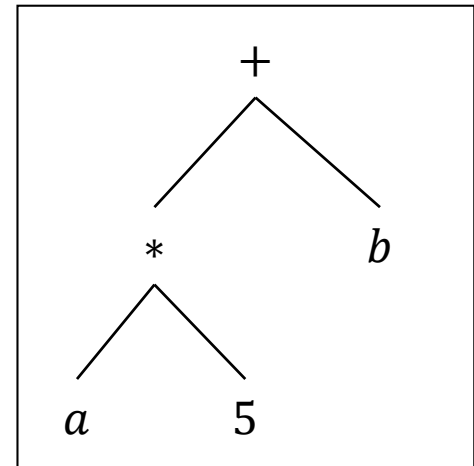
## 6.2.4 抽象语法树

### □ 构造抽象语法树的元操作

- `mknode(op, left, right)`: 建立一个运算符结点, 标号是`op`, 两个域`left`和`right`分别指向左子树和右子树;
- `mkleaf(id, entry)`: 建立一个标识符结点, 标号为`id`, 一个域`entry`指向标识符在符号表的入口;
- `mkleaf(num, val)`: 建立一个数结点, 标号为`num`, 一个域`val`用来存放数的值。

#### 【例6.10】 $a * 5 + b$ 的抽象语法树构造序列

- ① `p1 = mkleaf(a, entrya);`
- ② `p2 = mkleaf(5, 5);`
- ③ `p3 = mknode('*', p1, p2);`
- ④ `p4 = mkleaf(b, entryc);`
- ⑤ `p5 = mknode('+', p3, p4);`



## 6.2.4 抽象语法树

【例6.11】为表达式建立抽象语法树的属性文法

$$E \rightarrow E_1 + T \quad E.nptr = mknnode('+', E_1.nptr, T.nptr)$$
$$E \rightarrow E_1 - T \quad E.nptr = mknnode('-', E_1.nptr, T.nptr)$$
$$E \rightarrow T \quad E.nptr = T.nptr$$
$$T \rightarrow (E) \quad T.nptr = E.nptr$$
$$T \rightarrow id \quad T.nptr = mkleaf(id, id.entry)$$
$$T \rightarrow num \quad T.nptr = mkleaf(num, num.val)$$

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性



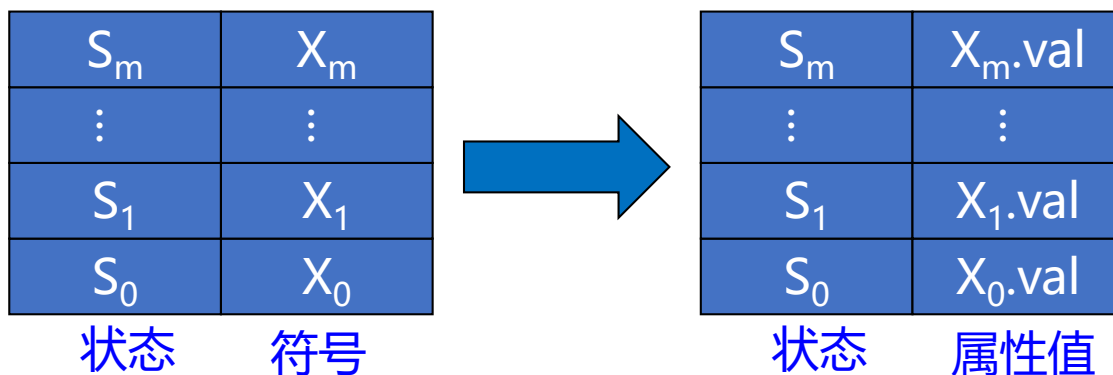
## 6.3 S-属性文法的自下而上计算

□ **S-属性文法**：只含有综合属性的文法。

- 综合属性可以在分析输入符号串的同时，由自下而上的分析器来计算；
- 分析器可以保存与栈中文法符号有关的综合属性，每当进行归约时，新的属性值就由栈中正在归约的产生式右边符号的属性值计算。

□ **S-属性文法**通常可以借助**LR分析器**实现。

- 分析器中附加一个域存放文法符号的属性值。



## 6.3 S-属性文法的自下而上计算

【例6.12】用LR分析器实现台式计算器

$L \rightarrow E$              $print(val[top])$     // top指栈顶

$E \rightarrow E_1 + T$      $val[ntop] = val[top - 2] + val[top]$  // ntop指归约后的新栈顶

$E \rightarrow T$

$T \rightarrow T_1 * F$      $val[ntop] = val[top - 2] * val[top]$

$T \rightarrow F$

$F \rightarrow (E)$          $val[ntop] = val[top - 1]$

$F \rightarrow i$

(0) $L \rightarrow E$	$print(val[top])$
(1) $E \rightarrow E + T$	$val[ntop] = val[top - 2] + val[top]$
(2) $E \rightarrow T$	
(3) $T \rightarrow T * F$	$val[ntop] = val[top - 2] * val[top]$
(4) $T \rightarrow F$	
(5) $F \rightarrow (E)$	$val[ntop] = val[top - 1]$
(6) $F \rightarrow i$	

状态	Action						Goto		
	i	+	*	(	)	#	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			

序号	状态栈	属性栈	输入串
1	0	#	3 * 5 + 4 #
2	05	#3	* 5 + 4 #
3	03	#3	* 5 + 4 #
4	02	#3	* 5 + 4 #
5	027	#3 *	5 + 4 #
6	0275	#3 * 5	+ 4 #
7	027 <u>10</u>	#3 * 5	+ 4 #
8	02	#15	+ 4 #
9	01	#15	+ 4 #
10	016	#15 +	4 #
11	0165	#15 + 4	#
12	0163	#15 + 4	#
13	0169	#15 + 4	#
14	01	#19	#
15	acc, 分析表里没有包含 $L \rightarrow E$		

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

## 6.4 L-属性文法和自顶向下翻译

□ **L-属性文法**：如果对于每个产生式 $A \rightarrow X_1 X_2 \dots X_n$ ，其语义规则中的每个属性或者是**综合属性**，或者是 $X_i (1 \leq i \leq n)$ 的一个**继承属性**且这个继承属性仅依赖于：

- 产生式**右部** $X_i$ 的**左边**符号 $X_1, X_2, \dots, X_{i-1}$ 的**属性**；
- 产生式**左部** $A$ 的**继承属性**。

### □ 说明

- **S-属性文法一定是L-属性文法**，因为L-属性文法定义中未对综合属性进行限制
- L-属性文法可以**一次遍历**就计算出所有属性值。
- 本节讨论L-属性文法的**自上向下翻译问题**，下一节讨论L-属性文法的**自下而上翻译问题**。

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

## 6.4.1 翻译模式

- 属性文法：为产生式配上语义动作。
- 翻译模式 (Translation Schemes)：和文法符号相关的属性和语义规则（语义动作），用花括号{}括起来，插入到产生式右部的合适位置上。

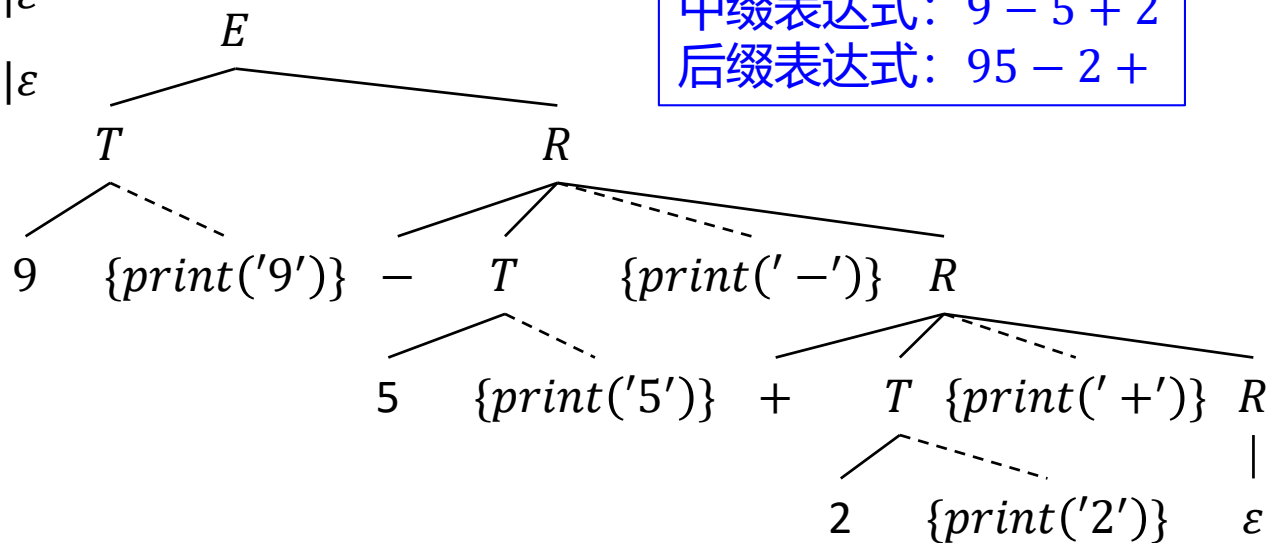
【例6.13】带加减的中缀表达式翻译成后缀表达式

$$E \rightarrow TR$$

$$R \rightarrow +T\{print(' +')\}R_1|\varepsilon$$

$$R \rightarrow -T\{print(' -')\}R_1|\varepsilon$$

$$T \rightarrow i\{print(i.val)\}$$



# 翻译模式设计

□ 只有综合属性：语义动作放到产生式右部末尾。

【例6.14】综合属性的翻译模式

$$T \rightarrow T_1 * F\{T.val = T_1.val * F.val\}$$



# 翻译模式设计

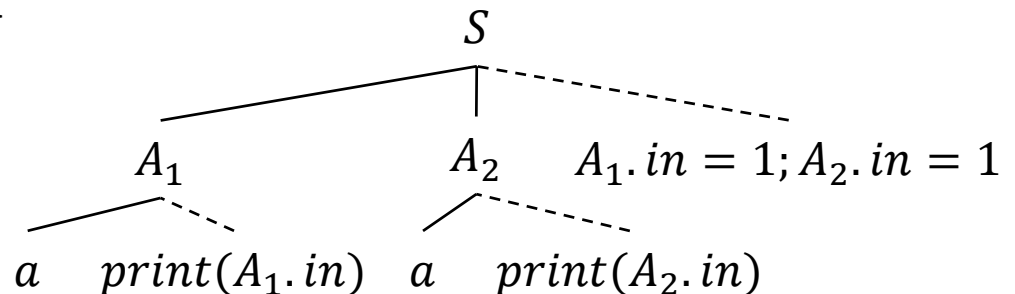
## □ 既有综合属性又有继承属性:

- ① 产生式右部符号的继承属性，必须在这个符号以前的动作中计算出来；
- ② 一个动作不能引用这个动作右边符号的综合属性；
- ③ 产生式左部的 $V_N$ 的综合属性，只有在它所引用的所有属性都计算出来以后才能计算（放到右边末尾）。

### 【例6.15】不满足条件①的翻译模式

$S \rightarrow A_1 A_2 \{A_1.in = 1; A_2.in = 1\}$

$A \rightarrow a \{print(A.in)\}$



# 翻译模式设计

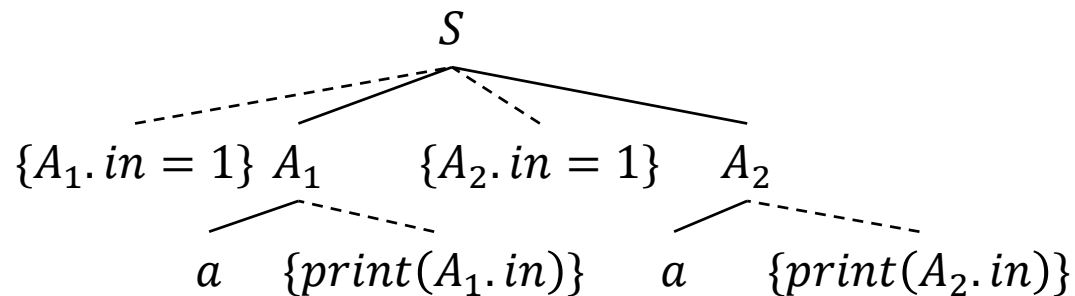
## □ 既有综合属性又有继承属性：

- ① 产生式右部符号的继承属性，必须在这个符号以前的动作中计算出来；
- ② 一个动作不能引用这个动作右边符号的综合属性；
- ③ 产生式左部的 $V_N$ 的综合属性，只有在它所引用的所有属性都计算出来以后才能计算（放到右边末尾）。

## 【例6.16】满足条件①②③的翻译模式

$$S \rightarrow \{A_1.in = 1\} A_1 \{A_2.in = 1\} A_2$$

$$A \rightarrow a \{print(A.in)\}$$



# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

# 消除翻译模式的左递归

## 带左递归的翻译模式

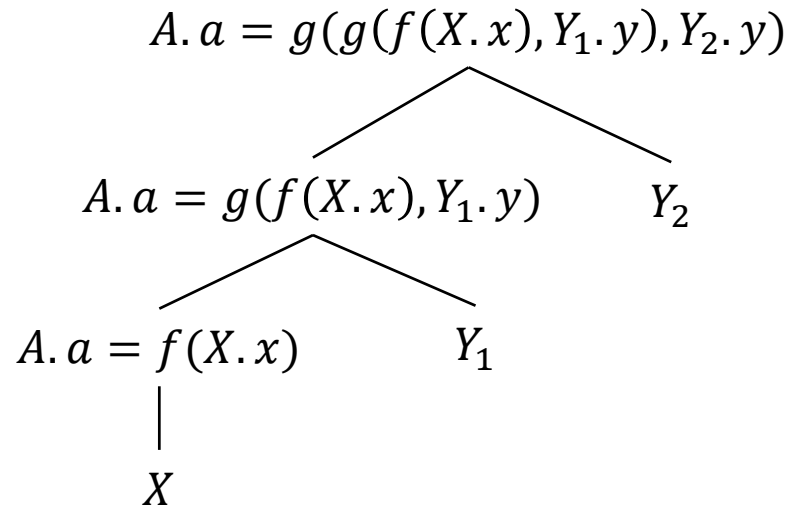
$$A \rightarrow A_1 Y \{A.a = g(A_1.a, Y.y)\}$$

$$A \rightarrow X \{A.a = f(X.x)\}$$

## 消除文法左递归

$$A \rightarrow XR$$

$$R \rightarrow YR | \varepsilon$$



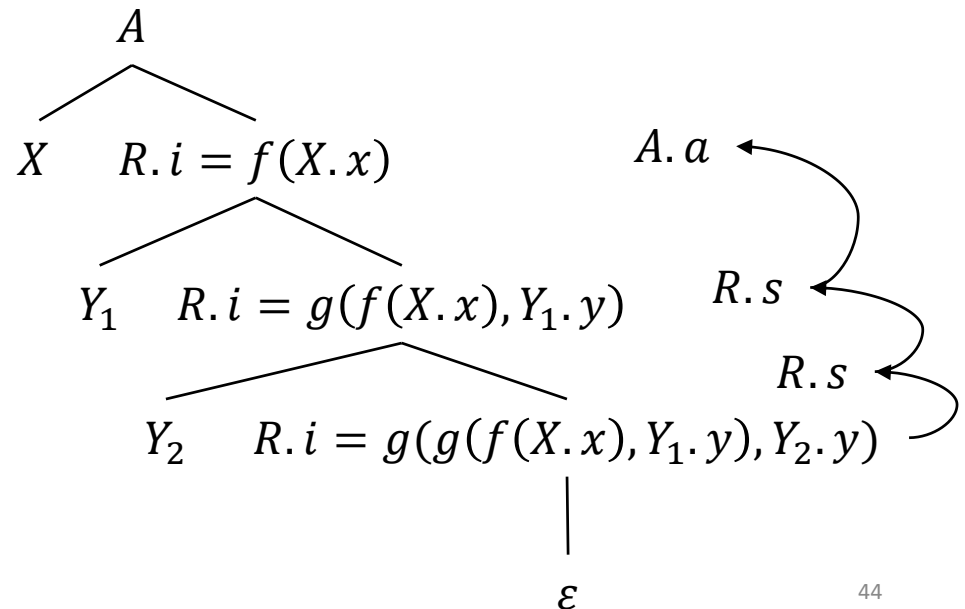
## 考虑语义动作的翻译模式

$$A \rightarrow X \{R.i = f(X.x)\} R \{A.a = R.s\}$$

$$R \rightarrow Y \{R_1.i = g(R.i, Y.y)\} R_1 \{R.s = R_1.s\}$$

$$R \rightarrow \varepsilon \{R.s = R.i\}$$

## 以XYR为例说明其等价性



# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

## 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作

□ 6.3节自下而上的翻译方法中，要求语义动作放在产生式末尾

➤ 解决办法：引入空符号产生式（只需要处理动作，不需要处理属性计算）。

$A \rightarrow \alpha\{Action\}\beta$ , 其中  $\beta \in (V_N \cup V_T)^*$ ,  $\beta \neq \varepsilon$

修改为：

①  $A \rightarrow \alpha M \beta$

②  $M \rightarrow \varepsilon\{Action\}$

## 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作

【例6.17】消除如下翻译模式中产生式中间的动作

$$E \rightarrow TR$$

$$R \rightarrow +T\{print(' +')\}R \mid -T\{print(' -')\}R \mid \varepsilon$$

$$T \rightarrow num\{print(num.val)\}$$

【解】

$$E \rightarrow TR$$

$$R \rightarrow +TMR \mid -TNR \mid \varepsilon$$

$$T \rightarrow num\{print(num.val)\}$$

$$M \rightarrow \varepsilon\{print(' +')\}$$

$$N \rightarrow \varepsilon\{print(' -')\}$$

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性



## 6.5.2 分析栈中的继承属性

### □ 【例6.18】 翻译模式

$$D \rightarrow T \{L.in = T.type\}$$
$$L$$
$$T \rightarrow int \{T.type = integer\}$$
$$T \rightarrow real \{T.type = real\}$$
$$L \rightarrow \quad \{L_1.in = L.in\}$$
$$L_1, id \{addType(id.entry, L.in)\}$$
$$L \rightarrow id \{addType(id.entry, L.in)\}$$

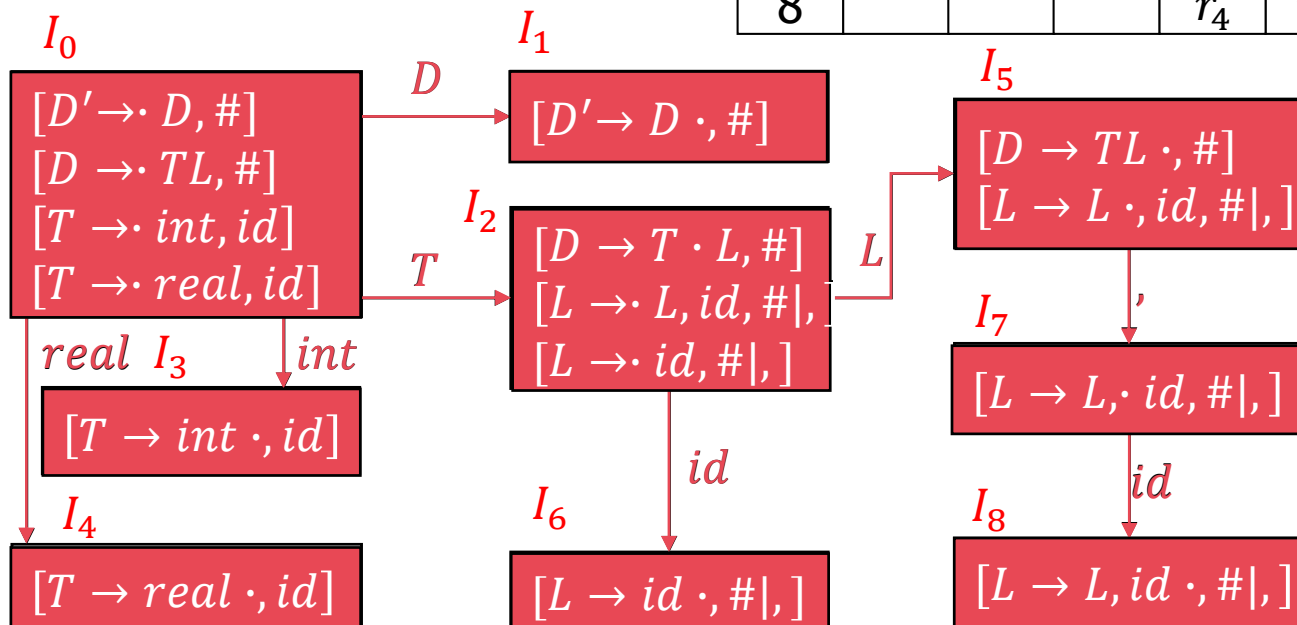
- (0)  $D' \rightarrow D$
- (1)  $D \rightarrow TL$
- (2)  $T \rightarrow int$
- (3)  $T \rightarrow real$
- (4)  $L \rightarrow L, id$
- (5)  $L \rightarrow id$

$First(D) = \{int, real\}$

$First(T) = \{int, real\}$

$First(L) = \{id\}$

	Action					Goto		
状态	int	real	id	,	#	D	T	L
0	$S_3$	$S_4$				1	2	
1					$acc$			
2			$S_6$					5
3			$r_2$					
4			$r_3$					
5				$S_7$	$r_1$			
6				$r_5$	$r_5$			
7			$S_8$					
8				$r_4$	$r_4$			



(1)  $D \rightarrow T \{L.in = T.type\}$

$L$

(2)  $T \rightarrow int \{T.type = integer\}$

(3)  $T \rightarrow real \{T.type = real\}$

(4)  $L \rightarrow \{L_1.in = L.in\}$

$L_1, id \{addType(id.entry, L.in)\}$

(5)  $L \rightarrow id \{addType(id.entry, L.in)\}$

	Action					Goto		
状态	int	real	id	,	#	D	T	L
0	$S_3$	$S_4$				1	2	
1					$acc$			
2			$S_6$					5
3			$r_2$					
4			$r_3$					
5				$S_7$	$r_1$			
6				$r_5$	$r_5$			
7			$S_8$					
8				$r_4$	$r_4$			

序号	状态栈	符号栈	输入串
1	0	#	$int\ p, q, r\ \#$
2	03	# $int$	$p, q, r\ \#$
3	02	# $T$	$p, q, r\ \#$
4	026	# $TP$	$, q, r\ \#$
5	025	# $TL$	$, q, r\ \#$
6	0257	# $TL,$	$q, r\ \#$
7	02578	# $TL, q$	$, r\ \#$
8	025	# $TL$	$, r\ \#$

序号	状态栈	符号栈	输入串
9	0257	# $TL,$	$r\ \#$
10	02578	# $TL, r$	$\#$
11	025	# $TL$	$\#$
12	01	# $D$	$\#$
13	01	$acc$	$\#$

(1)  $D \rightarrow T \{L.in = T.type\}$

$L$

(2)  $T \rightarrow int \{T.type = integer\}$

(3)  $T \rightarrow real \{T.type = real\}$

(4)  $L \rightarrow \{L_1.in = L.in\}$

$L_1, id \{addType(id.entry, L.in)\}$

(5)  $L \rightarrow id \{addType(id.entry, L.in)\}$

序号	状态栈	符号栈	输入串
1	0	#	$int\ p, q, r\ \#$
2	03	$\#int$	$p, q, r\ \#$
3	02	$\#T$	$p, q, r\ \#$
4	026	$\#Tp$	$, q, r\ \#$
5	025	$\#TL$	$, q, r\ \#$
6	0257	$\#TL,$	$q, r\ \#$
7	02578	$\#TL, q$	$, r\ \#$
8	025	$\#TL$	$, r\ \#$

## 【方案1】使用L存储属性

①  $D \rightarrow TL$

②  $T \rightarrow int \{val[ntop] = integer\}$

③  $T \rightarrow real \{val[ntop] = real\}$

④  $L \rightarrow L_1, id \{addType(val[top], val[top - 2]); val[ntop] = val[top - 2]\}$

⑤  $L \rightarrow id \{addType(val[top], val[top - 1]); val[ntop] = val[top - 1]\}$

序号	状态栈	符号栈	输入串
9	0257	$\#TL,$	$r\ \#$
10	02578	$\#TL, r$	$\#$
11	025	$\#TL$	$\#$
12	01	$\#D$	$\#$
13	01	$acc$	$\#$

(1)  $D \rightarrow T \{L.in = T.type\}$

$L$

(2)  $T \rightarrow int \{T.type = integer\}$

(3)  $T \rightarrow real \{T.type = real\}$

(4)  $L \rightarrow \{L_1.in = L.in\}$

$L_1, id \{addType(id.entry, L.in)\}$

(5)  $L \rightarrow id \{addType(id.entry, L.in)\}$

## 【方案2】不使用L存储属性

①  $D \rightarrow TL$

②  $T \rightarrow int \{val[ntop] = integer\}$

③  $T \rightarrow real \{val[ntop] = real\}$

④  $L \rightarrow L_1, id \{addType(val[top], val[top - 3])\}$

⑤  $L \rightarrow id \{addType(val[top], val[top - 1])\}$

序号	状态栈	符号栈	输入串
1	0	#	$int\ p, q, r\ \#$
2	03	# $int$	$p, q, r\ \#$
3	02	# $T$	$p, q, r\ \#$
4	026	# $Tp$	$, q, r\ \#$
5	025	# $TL$	$, q, r\ \#$
6	0257	# $TL,$	$q, r\ \#$
7	02578	# $TL, q$	$, r\ \#$
8	025	# $TL$	$, r\ \#$

序号	状态栈	符号栈	输入串
9	0257	# $TL,$	$r\ \#$
10	02578	# $TL, r$	$\#$
11	025	# $TL$	$\#$
12	01	# $D$	$\#$
13	01	$acc$	$\#$

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性

## 6.5.3 模拟继承属性的计算

□ 只有根据文法**预知**属性值在栈中**存放位置**时，才能有效地在分析栈中处理属性值。

➤ 有时位置可能存在冲突。

**【例6.19】**  $A$ 和 $C$ 之间可能有 $B$ 也可能没有 $B$ ，当通过 $C \rightarrow c$ 进行归约时， $C.i$ 可能在 $val[top - 1]$ 处，也可能在 $val[top - 2]$ 处：

$$S \rightarrow aAC \{C.i = A.s\}$$

$$S \rightarrow bABC \{C.i = A.s\}$$

$$C \rightarrow c \{C.s = g(C.i)\}$$

**【修改】**  $M \rightarrow \varepsilon$ 时从 $val[top - 1]$ 处取到 $A.s$ ， $C \rightarrow c$ 归约时， $C.i$ 总在 $val[top - 1]$ 处。

$$S \rightarrow aAC \{C.i = A.s\}$$

$$S \rightarrow bABMC \{M.i = A.s; C.i = M.s\}$$

$$C \rightarrow c \{C.s = g(C.i)\}$$

$$M \rightarrow \varepsilon \{M.s = M.i\}$$

# 第六章 属性文法和语法制导翻译

## □ 6.1 属性文法

## □ 6.2 基于属性文法的处理方法

- 6.2.1 依赖图
- 6.2.2 树遍历的属性计算方法
- 6.2.3 一遍扫描的处理方法
- 6.2.4 抽象语法树

## □ 6.3 S-属性文法的自下而上计算

## □ 6.4 L-属性文法和自顶向下翻译

- 6.4.1 翻译模式
- 6.4.2 自顶向下翻译

## □ 6.5 自下而上计算继承属性

- 6.5.1 从翻译模式中去掉嵌入在产生式中间的动作
- 6.5.2 分析栈中的继承属性
- 6.5.3 模拟继承属性的计算
- 6.5.4 用综合属性代替继承属性



## 6.5.4 用综合属性代替继承属性

□ 改变基础文法可以避免继承属性。

【例6.20】Pascal说明语句如 $m, n:integer$ ，标识符由 $L$ 产生类型，但类型不在 $L$ 子树中：

$$D \rightarrow L:T$$

$$T \rightarrow integer \mid char$$

$$L \rightarrow L, id \mid id$$

【修改】 重构文法，使类型作为标识符表的最后一个元素：

$$D \rightarrow id L$$

$$L \rightarrow , id L \mid : T$$

$$T \rightarrow integer \mid char$$

# 第六章作业

【作业6-1】文法 $G[E]$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow num.num \mid num$$

- (1) 给出确定每个子表达式结果类型的属性文法。
- (2) 扩充(1)的属性文法，使其把表达式翻译成后缀式，同时也能确定结果类型。注意实数和整数相加得实数，应保证后缀式中两个加数是同型的，可以采用 `int2real` 把整型转为实型。