



# 编译原理

## 第六章 属性文法和语法制导翻译

# 第六章 属性文法和语法制导翻译

- 属性文法
- 基于属性文法的处理方法
- S- 属性文法的自下而上计算
- L- 属性文法和自顶向下翻译

# 第六章 属性文法和语法制导翻译

- 属性文法
- 基于属性文法的处理方法
- S- 属性文法的自下而上计算
- L- 属性文法和自顶向下翻译

## 6.2 基于属性文法的处理方法

输入串  $\rightarrow$  语法树  $\rightarrow$  按照语义规则计算属性

- 由源程序的语法结构所驱动的处理办法就是**语法制导翻译法**
- 语义规则的计算
  - 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
- 对输入符号串的**翻译**也就是根据语义规则进行**计算**的结果

## 6.2 基于属性文法的处理方法

- 依赖图
- 树遍历
- 一遍扫描

## 6.2 基于属性文法的处理方法

- 依赖图
- 树遍历
- 一遍扫描


# 依赖图

- 在一棵语法树中的结点的继承属性和综合属性之间的相互依赖关系可以由**依赖图**（有向图）来描述
- 为每一个包含过程调用的语义规则引入一个**虚综合属性  $b$** ，这样把每一个语义规则都写成

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

的形式

- 依赖图中为每一个属性设置一个结点，如果属性  $b$  依赖于属性  $c$ ，则从属性  $c$  的结点有一条有向边连到属性  $b$  的结点。

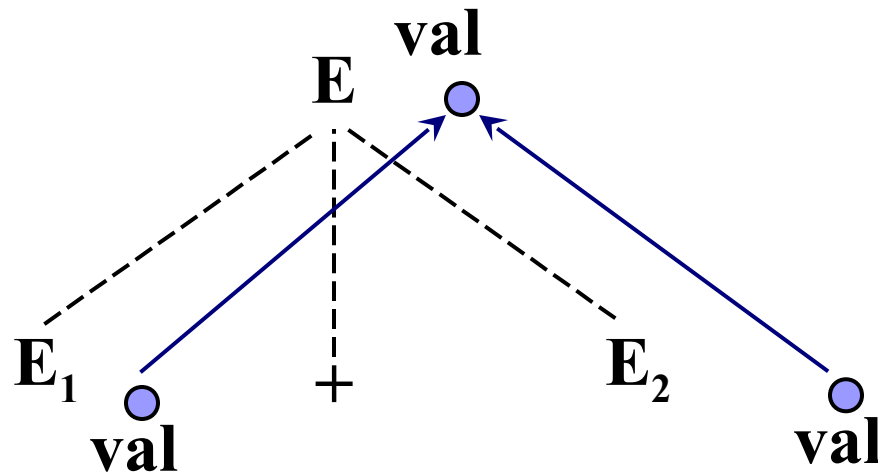


```
for 语法树中每一结点 n do
    for 结点 n 的文法符号的每一个属性 a do
        为 a 在依赖图中建立一个结点;
for 语法树中每一个结点 n do
    for 结点 n 所用产生式对应的每一个语义规则
         $b := f(c_1, c_2, \dots, c_k)$  do
        for  $i := 1$  to  $k$  do
            从  $c_i$  结点到  $b$  结点构造一条有向边;
```



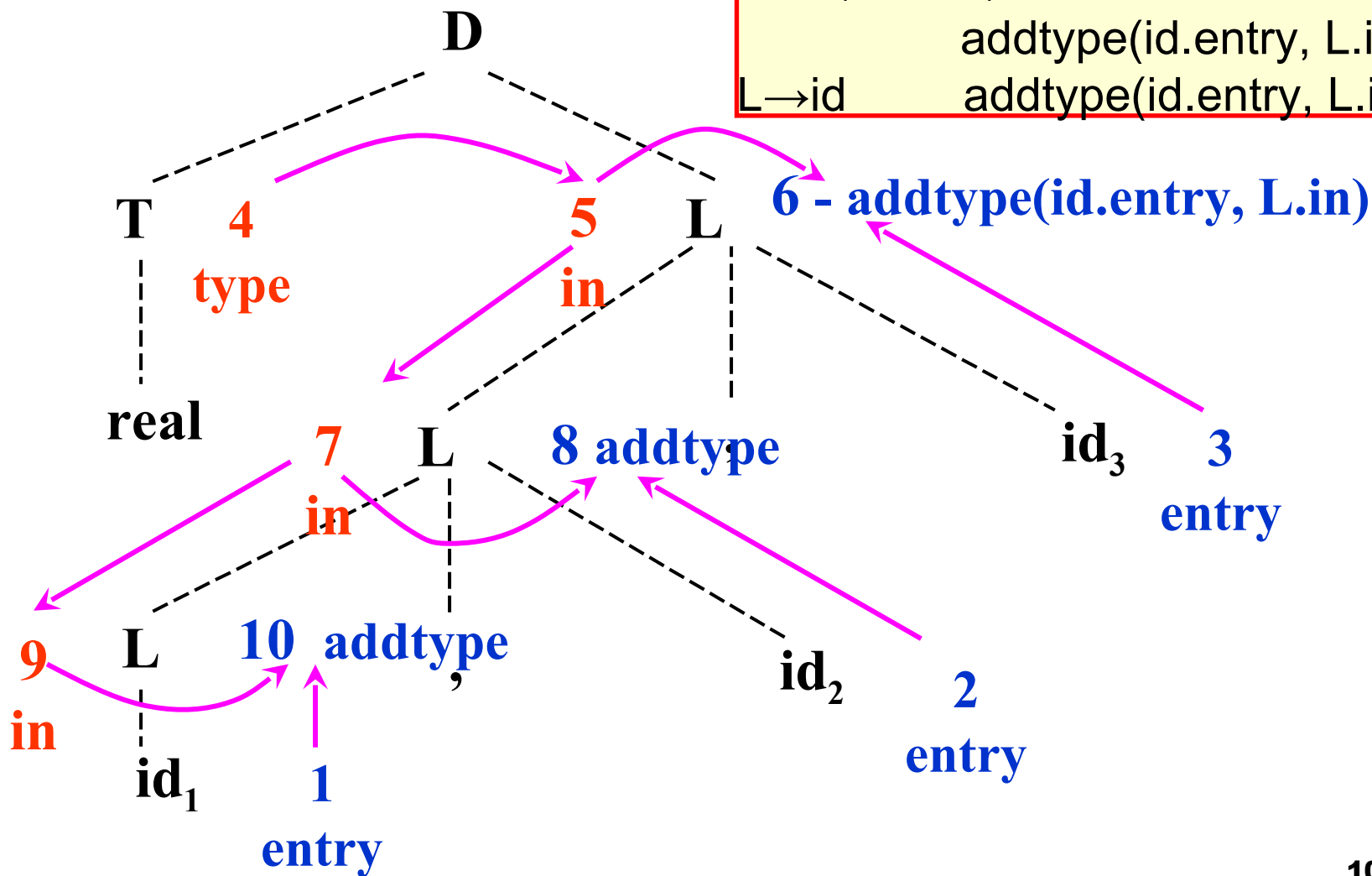
$$E \rightarrow E_1 + E_2$$

$$E.val := E_1.val + E_2.val$$



产生式	语义规则
$D \rightarrow TL$	$L.in := T.type$
$T \rightarrow int$	$T.type := integer$
$T \rightarrow real$	$T.type := real$
$L \rightarrow L_1, id$	$L_1.in := L.in$
	$addtype(id.entry, L.in)$
$L \rightarrow id$	$addtype(id.entry, L.in)$

句子  $real\ id_1,\ id_2,\ id_3$   
的带注释的语法树的依赖图



# 良定义的属性文法

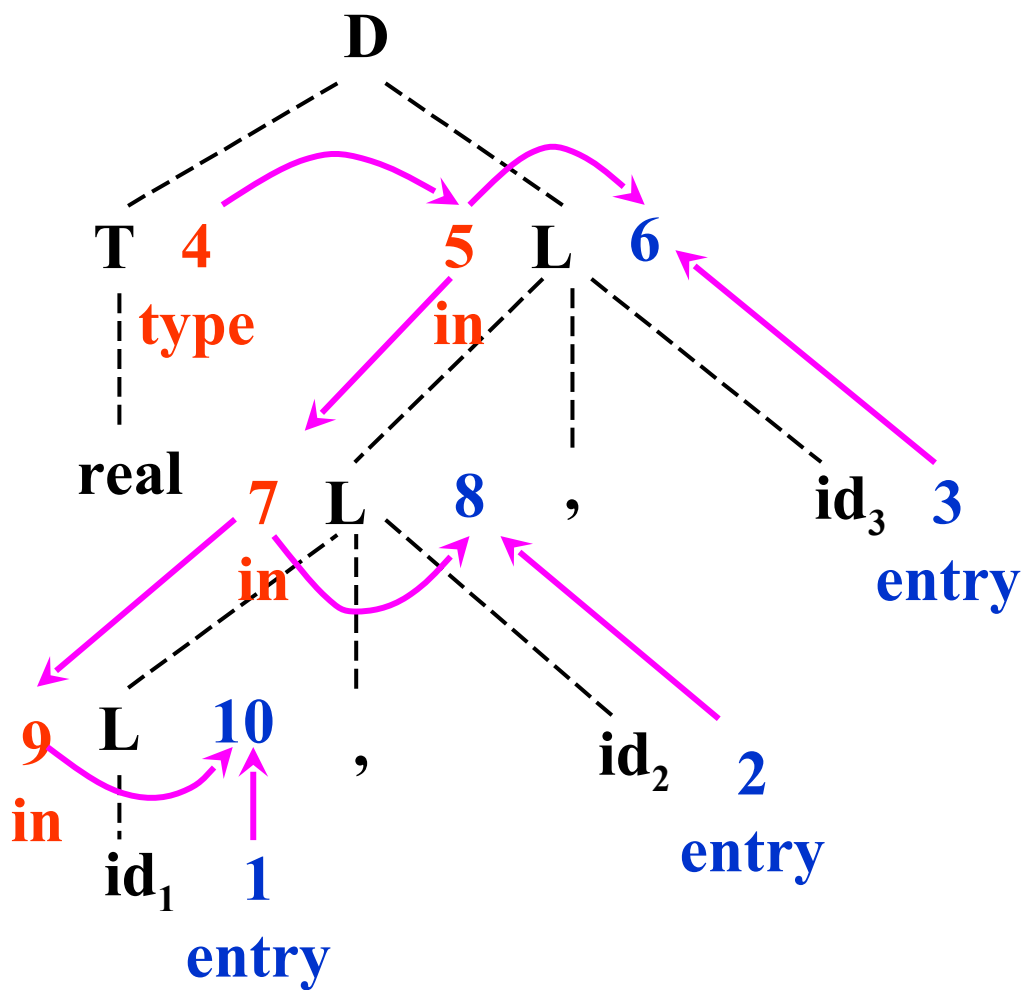
- 如果一属性文法不存在属性之间的循环依赖关系，那么称该文法为良定义的

# 属性的计算次序

- 一个依赖图的任何拓扑排序都给出一个语法树中结点的语义规则计算的有效顺序
- 属性文法说明的翻译是很精确的
  - 基础文法用于建立输入符号串的语法分析树
  - 根据语义规则建立依赖图
  - 从依赖图的拓扑排序中，我们可以得到计算语义规则的顺序

输入串 → 语法树 → 依赖图 → 语义规则计算次序

# 句子 $\text{real id}_1, \text{id}_2, \text{id}_3$ 带注释的语法树的依赖图



```

a4 := real;
a5 := a4
addtype (id3.entry, a5);
a7 := a5;
addtype (id2.entry, a7);
a9 := a7
addtype (id1.entry, a9);
    
```

## 6.2 基于属性文法的的处理方法

- 依赖图
- 树遍历
- 一遍扫描

## 6.2.2 树遍历的属性计算方法

- 通过树遍历的方法计算属性的值
  - 假设语法树已建立，且树中已带有开始符号的继承属性和终结符的综合属性
  - 以某种次序遍历语法树，直至计算出所有属性
    - 深度优先，从左到右的遍历

## ■ 计算思维的典型方法 -- 递归

- 问题的解决又依赖于类似问题的解决，只不过后者的复杂程度或规模较原来的问题更小
- 一旦将问题的复杂程度和规模化简到足够小时，问题的解法其实非常简单

While 还有未被计算  
VisitNode(S) /

procedure VisitNode  
begin

if N 是一个非终结符

/\* 假设它的产生式为  $N \rightarrow X_1 \cdots X_m$  \*/

for i := 1 to m do

if  $X_i \in V_N$  then /\* 即  $X_i$  是非终结符 \*/

begin

计算  $X_i$  的所有能够计算的继承属性;

VisitNode ( $X_i$ )

end;

计算 N 的所有能够计算的综合属性

end



例 考虑属性的文法  $G$ 。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  $g$

产生式

$S \rightarrow XYZ$

$X \rightarrow x$

$Y \rightarrow y$

$Z \rightarrow z$

语义规则

$Z.h := S.a$

$X.c := Z.g$

$S.b := X.d - 2$

$Y.e := S.b$

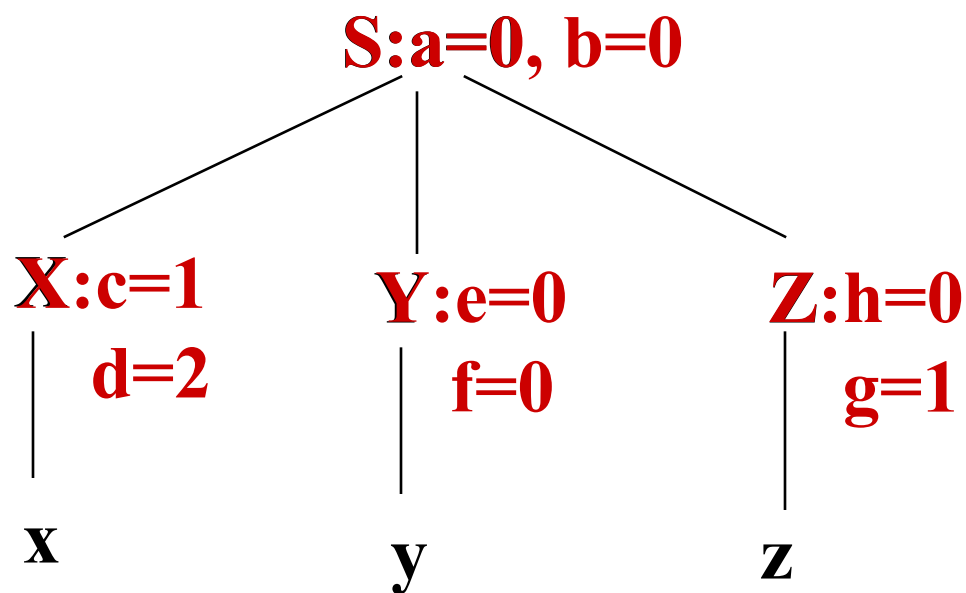
$X.d := 2 * X.c$

$Y.f := Y.e * 3$

$Z.g := Z.h + 1$

假设  $S.a$  的初始值为 0，输入串为

产生式	语义规则
$S \rightarrow XYZ$	$Z.h := S.a$
	$X.c := Z.g$
	$S.b := X.d - 2$
	$Y.e := S.b$
$X \rightarrow x$	$X.d := 2 * X.c$
$Y \rightarrow y$	$Y.f := Y.e * 3$
$Z \rightarrow z$	$Z.g := Z.h + 1$



# 小结

- 基于属性文法的处理方法
  - 依赖图
  - 树遍历