编译原理

2020年5月

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

编译原理
第五章

上海交通大学

张冬茉

Email:zhang-dm@cs.sjtu.edu.cn

2020年5月

第5章 语法制导的翻译

经过词法分析、语法分析后，源程序在静态结构上的正确性得到了保证，编译程序接着将语义信息和程序设计语言的结构联系起来，进行语义计算。使用上下文无关文法来引导对语言的翻译。

本章的重点：

- 介绍有关语义分析及翻译问题
- 语义描述和语义处理的方法：属性文法和语法制导翻译法
- 基于属性文法的处理方法及属性的计算

5.1 语法制导定义

- 语法制导定义是关于语言翻译的高层次规格说明，用于描述语义计算的规则。它隐蔽了许多具体实现细节。（做什么）

翻译模式则指明了语义规则的计算顺序，以便说明某些实现细节。(怎么做)

- 语法制导定义（Syntax-Directed-Definition,SDD):

使用上下文无关文法来说明输入的语法结构。它通过每个文法符号和一个属性集合相关联，通过每一个产生式和一个语义规则集合相关联。语义规则用来计算与产生式中出现的符号相关联的属性的值。文法和语义规则集合构成了语法制导定义。

属性文法

- 属性文法又称属性翻译文法，是对上下文无关文法中的每个文法符号配备相关的属性（语义值）。例如：类型，数值，代码序号，符号表的内容。
- 属性与变量一样，可以进行计算和传递。
- 属性加工的过程即为语义处理的过程。
- 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。

属性分类

属性通常分为：综合属性和继承属性

- 综合属性值是通过分析树中对应该文法符号的节点和其子节点的属性值计算出来的，综合属性通常用于“自下而上”传递信息。
- 继承属性值则是由该节点和其兄弟节点及父节点的属性值计算出来的，继承属性通常用于“自上而下”传递信息。
- 为产生式A→α 配备的语义规则形式为：

  b:=f(c1,c2，… ,ck)

  其中：f 是一个函数，

  （1）如果c1,c2，… ,ck是α中的文法符号的属性，则b为A的综合属性

  （2）如果c1,c2，… ,ck是A和α中的文法符号的属性，则b为α中的某个文法符号继承属性

上述两种情况，都称属性b依赖于属性 c1,c2，… ,ck

属性计算规则：

(1) 终结符只有综合属性，且由词法分析器提供;

(2) 非终结符既可以由综合属性，也可以有继承属性，文法的开始符的继承属性不能计算，只能有初始值;

(3) 必须提供一条计算规则的属性:

产生式左部的综合属性，产生式右部的继承属性

(4) 属性的计算规则中只能使用本条产生式中的文法符号的属性，其中：产生式左部的继承属性和产生式右部的综合属性只能被使用，它们来自于其它产生式的属性计算

(5) 语义规则描述的工作包括:

属性计算，静态语义检查，符号表操作，中间代码生成等

例5.1

考虑非终结符A,B和C,其中：A有一个继承属性a和一个综合属性b，B有一个综合属性 c, C有一个继承属性d。

产生式A→BC可能有规则:

C.d:=B.c+1

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

而属性A.a和B.c在其他地方计算

综合属性

在语法树中，一个结点的综合属性的值由其子结点的属性值确定。因此，通常使用自底向上的方法在每个结点处使用语义规则计算综合属性的值。

仅仅包含综合属性的SDD称为S属性的SDD。

仅仅使用综合属性的文法称为S-属性文法。

S—属性文法的SDD

例如：一个台式计算器程序的语法制导定义如图5-1

PRODUCTION	SEMANTIC RULES
$L \rightarrow E\ \mathbf{n}$	$print(E.val)$
$E \rightarrow E_1 + T$	$E.val := E_1.val + T.val$
$E \rightarrow T$	$E.val := T.val$
$T \rightarrow T_1 * F$	$T.val := T_1.val \times F.val$
$T \rightarrow F$	$T.val := F.val$
$F \rightarrow (\ E\ )$	$F.val := E.val$
$F \rightarrow \mathbf{digit}$	$F.val := \mathbf{digit}.lexval$

图5-1 一个台式计算器程序的语法制导定义

继承属性

在语法树中，一个结点的继承属性的值由此结点的父结点和/或兄弟结点的某些属性值确定。

用继承属性来表示程序设计语言结构中的上下文依赖关系很方便

因此，通常使用自上而下的方法在每个结点处使用语义规则计算继承属性的值。

包含综合属性和特定继承属性的SDD称为L-属性的SDD。

L—属性文法的SDD

一个语法制导定义是L属性定义，如果对每个产生式：

$$A \rightarrow X1\ X2\ \ldots\ Xn,$$

其右部符号 Xj（I≤j≤n）的每个继承属性仅依赖于下列属性：

（1）产生式中Xj 左边的符号 X1，X2，…，Xj-1的属性。

（2） A 的继承属性

。

- L属性定义所规定的这些限制确保一个动作不会引用一个还没有计算出来的属性。
- 注意，每个 S 属性定义都是L属性定义，因为1和2只限制了继承属性。

L—属性文法的SDD

例如：一个简单类型声明的L—**属性文法的**语法制导定义如图5-2

PRODUCTION	SEMANTIC RULES
$D \rightarrow T\ L$	$L.in := T.type$
$T \rightarrow \mathbf{int}$	$T.type := integer$
$T \rightarrow \mathbf{real}$	$T.type := real$
$L \rightarrow L_1\ ,\ \mathbf{id}$	$L_1.in := L.in$ $addtype(\mathbf{id}.entry,\ L.in)$
$L \rightarrow \mathbf{id}$	$addtype(\mathbf{id}.entry,\ L.in)$

图5-2 一个简单类型声明的L—属性文法的语法制导定义

在语法分析树的结点上对SDD求值

- 注释分析树：一个标注了各结点的属性值的语法分析树
- 例：一个台式计算器程序的语法制导定义 实现的3*5+4n的注释分析树如图5-3

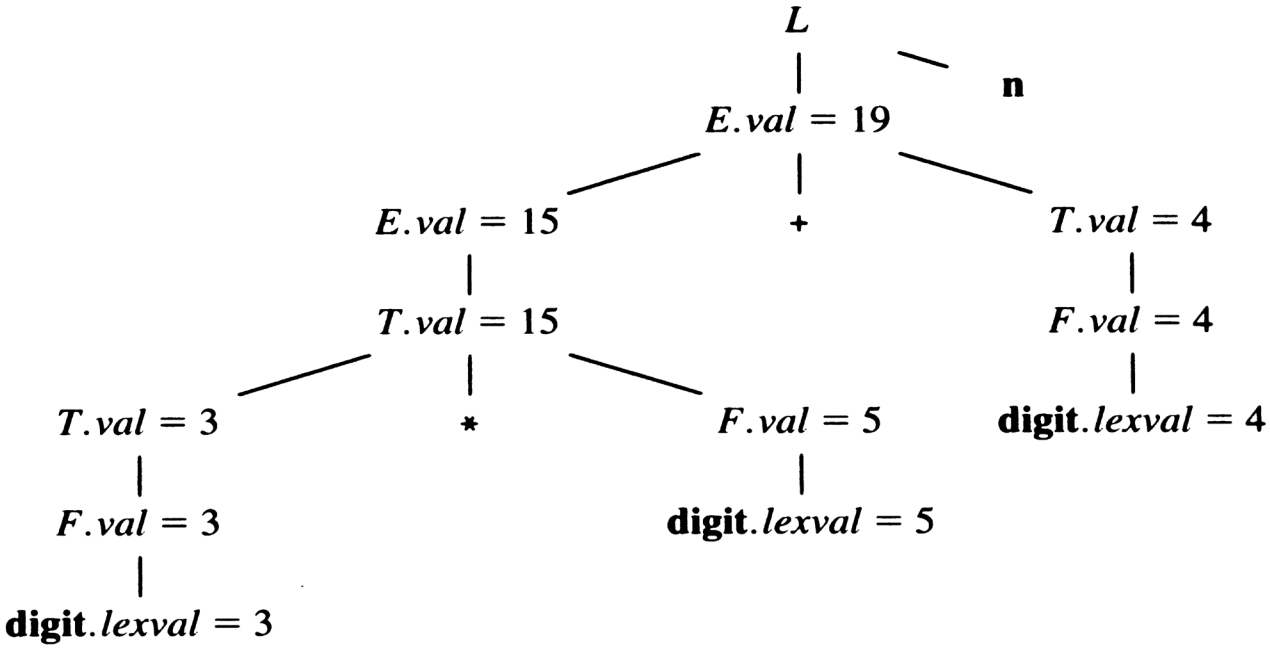

图5-3 3*5+4n的注释分析树

- 例：一个台式计算器程序的语法制导定义 实现的语句 real id1，id2，id3 的注释分析树如图5-4

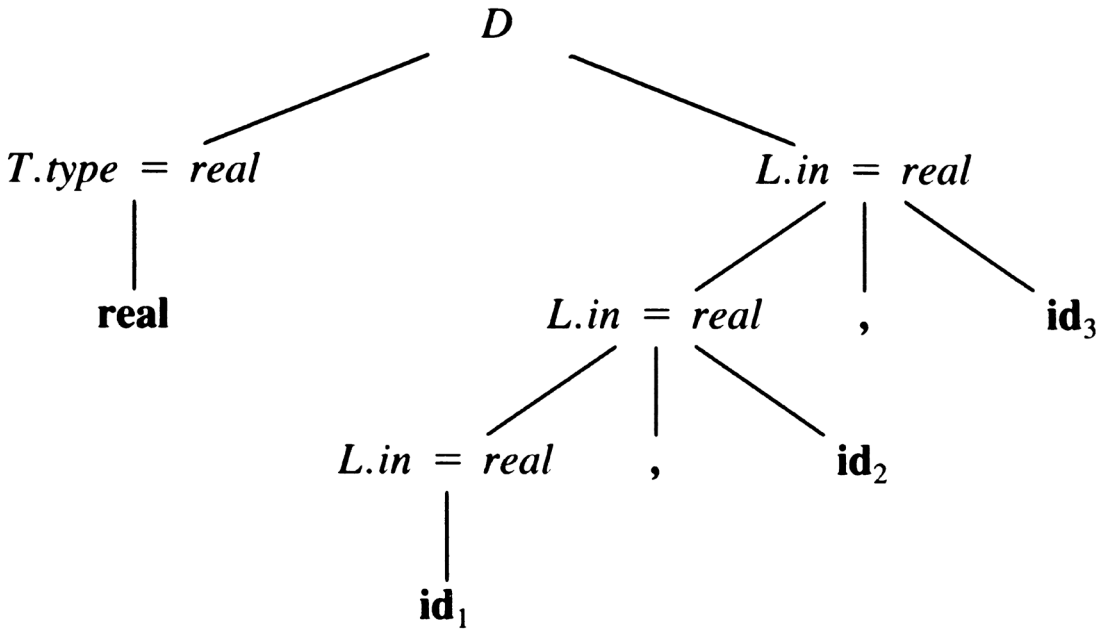

图5-4 real id1，id2，id3的注释分析树

在语法分析树的结点上对SDD求值

- 注释分析树中结点的属性值的构造原则：必须先求出该属性值所依赖的所有属性值。
- S—属性文法的结点属性值必须先求出该结点的所有子结点的属性值
- 具有综合属性和继承属性的SDD，不能保证有一个顺序来对各结点上的属性进行求值，
- 例如：考虑非终结符A和B,其中：A.s有是一个综合属性，B.i是一个继承属性，

  产生式A→B和规则如下： A.s :=B.i

  B.i:=A.s+1

  这样的规则就是循环定义。

- 解决的方法：用L—属性文法

5.2 SDD的求值顺序

- **依赖图：**是一个有向图。其中每个节点表示一个属性，边表示属性间的依赖关系，如果属性 b 依赖于属性 c，那么从 c 到 b 就有一条有向边。
- 例：产生式：　　$E \rightarrow E_1 + E_2$

  对应的语义规则为：　$E.val=E_1.val$和$E_2.val$

依赖图如图 5-5

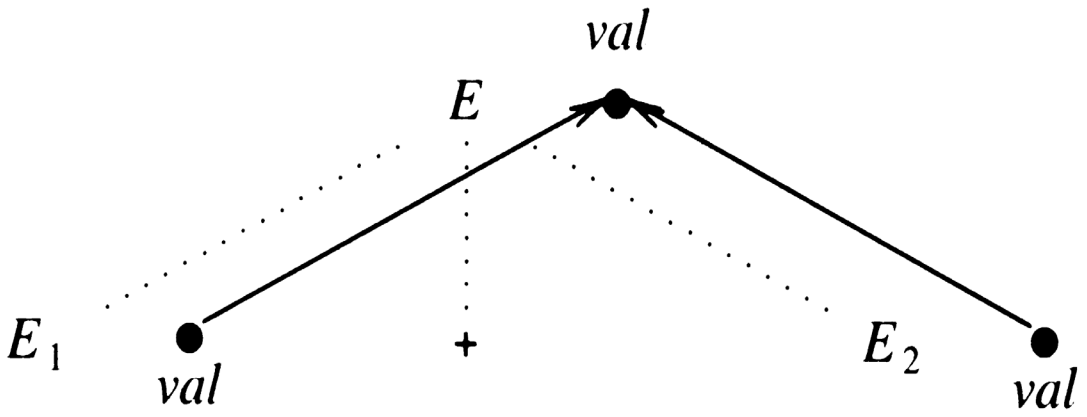

图5-5 $E.val$是$E_1.val$和$E_2.val$的综合

依赖图的构造方法

给定一棵分析树，其依赖图是按照下面的步骤构造出来的。

- for 分析树的每个节点 n do
- for 与节点n 对应的文法符号的每个属性 a do
- 在依赖图中为 a 构造一个节点;
- for 分析树的每个节点 n do
- for 节点 n 所用产生式对应的每条语义规则 b := f (c1, c2 ,…,ck) do
- for i := 1 to k do
- 从节点 ci 到节点 b 构造一条有向边;

例5.2

- 在图5-2的语法制导定义中：
- 非终结符 D 所产生的声明是由关键字 int 或 real 后跟一个标识符表所组成的。
- 非终结符 T 有一个综合属性 type，其值由声明中的关键字来确定。
- 产生式 D→TL 相关联的语义规则： L.in := T.type 用来将继承属性 L.in 置为所声明的类型。语义规则使用继承属性 L.in 把该类型信息沿分析树向下传递。
- L 产生式相关联的语义规则调用过程 addtype，该过程将各标识符的类型填加到符号表的相应表项中（终结符只有综合属性，且由词法分析器提供的属性 entry 指向）。

PRODUCTION	SEMANTIC RULES
$D \rightarrow T\ L$	$L.in := T.type$
$T \rightarrow \mathbf{int}$	$T.type := integer$
$T \rightarrow \mathbf{real}$	$T.type := real$
$L \rightarrow L_1\ ,\ \mathbf{id}$	$L_1.in := L.in$ $addtype(\mathbf{id}.entry,\ L.in)$
$L \rightarrow \mathbf{id}$	$addtype(\mathbf{id}.entry,\ L.in)$

图5-2 一个简单类型声明的L—属性文法的语法制导定义

图5-4给出的语句 real id1，id2，id3 的注释分析树对应的依赖图如图5-6。

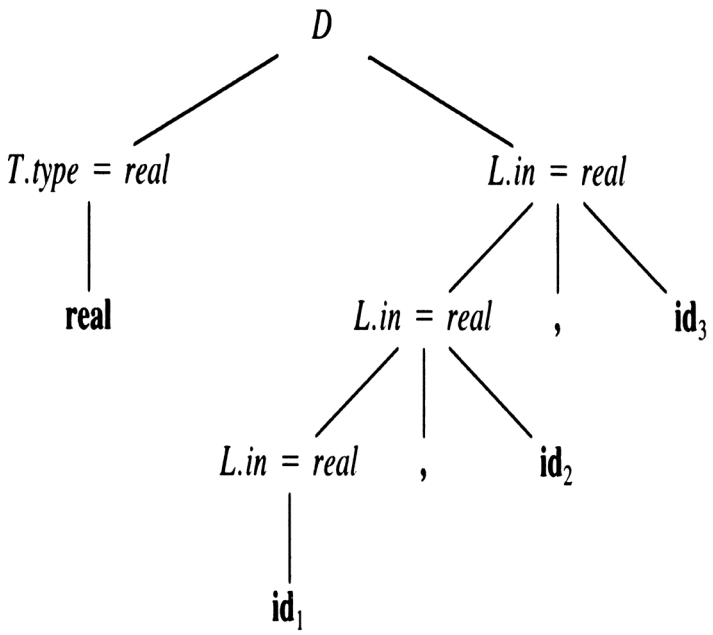

图5-4 real id1，id2，id3的注释分析树

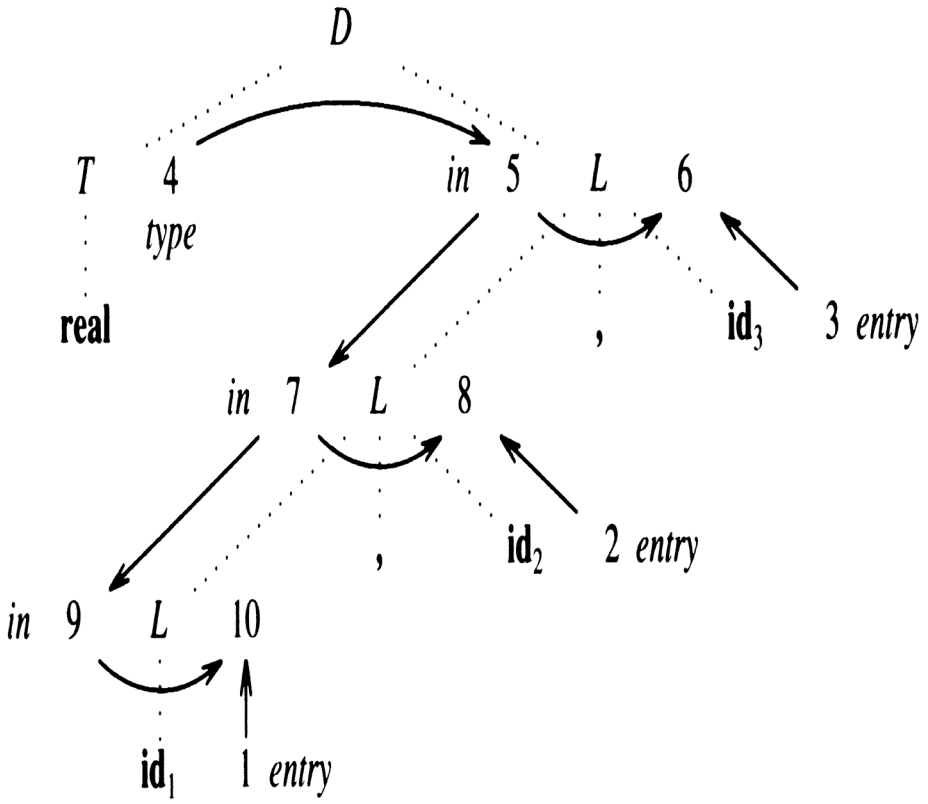

图5-6 图5-4中分析树的依赖图

计算顺序

依赖图中的节点用数字标记，这些数字将在后面用到。从节点4的 T.type 到节点5的 L.in 有一条边，这是因为根据产生式 D→TL 的语义规则 L.in := T.type，继承属性 L.in 依赖于综合属性 T.type。因为产生式L→L1，id 具有语义规则 L1.in := L.in，从而导致 L1.in 依赖于 L.in，所以有两条向下的边进入节点7和9。每个与 L 产生式相关联的语义规则 addtype (id.entry，L.in) 都产生一个虚属性，节点6，8和10都是为这些虚属性建立的。

从依赖图中我们可以得到语义规则的计算顺序。 ---依赖图的任何拓扑排序都给出了一个分析树中各节点语义规则计算的正确顺序。

例5.3

在图5-6的依赖图中，每条边都是从编号小的节点指向编号大的节点，所以将这些节点按编号从小到大写出即可得到该依赖图的一种拓扑排序。从该拓扑排序可以得到下面的翻译程序（其中 a_n表示依赖图中编号为 n 的节点的属性）：

(1) a_4:=real;

(2) a_5:=a_4;

(3) addtype(id_3.entry,a_5)

(4) a_7:=a_5;

(5) addtype(id_2.entry,a_7)

(6) a_9=a_7;

(7) addtype(id_1.entry,a_9)

具有受控副作用的语义规则

语义规则描述的工作包括:

属性计算，静态语义检查，符号表操作，中间代码生成等。

其中，属性计算的方法在语义规则中的描述可以采用综合属性和继承属性的计算方法，而静态语义检查，符号表操作，中间代码生成等工作则需要被称为具有受控副作用的语义规则来描述。

具有受控副作用的SDD需要满足以下两个条件之一:

(1) 支持那些不会对属性求值产生约束的附带的副作用

例如: 一个台式计算器程序的语法制导定义中的产生式:

L → En 对应的语义规则为: print(E.val)

为L定义了一个“虚综合属性”（打印 E 所产生的算术表达式的值）

具有受控副作用的语义规则

(2) 对允许的求值顺序添加约束，使得以任何允许的顺序求值都会产生相同的翻译结果

例如：一个简单类型声明的L—属性文法的语法制导定义中的产生式：

L→ L1 , id 对应的语义规则中的： addtype (id.entry，L.in)

为调用过程 addtype，该过程将各标识符的类型填加到符号表的相应表项中（由属性 entry 指向）产生一个“虚综合属性”。

//5.3语法制导翻译的应用

按语法分析顺序执行语义规则构造只具有附加语义属性值的结点分析树的方法，这种分析树为抽象语法树，分别采用：

- 基于LR分析法且SDD为S-属性的；
- 基于LL分析法且SDD为L-属性的；

//5.4语法制导的翻译方案

语法制导的翻译（syntax-directed translation SDT）方案是对语法制导定义的一种补充，是在产生式右部嵌入程序片段（语义动作）的一个上下文无关文法，通常SDT是按语法分析过程中实现的，不会实际构造一棵语法分析树，适用于：

- 基于LR分析法且SDD为S-属性文法；
- 基于LL分析法且SDD为L-属性文法；

注意：第六章中间代码生成将采用SDT方法

第五章 总结

主要内容：

- 程序语言的语法制导定义方法；
- 属性文法，综合属性和继承属性；
- 语义规则；
- S-属性文法和L-属性文法；
- 注释分析树与依赖树；
- 语法制导翻译的基本方法；

课后要求：

- 完成课后作业；
- 简单阅读教材P191-215内容，该部分的具体内容将在第六章进行详细讲解

谢 谢!