第五章　语法制导翻译技术

语法制导翻译技术是实现语义分析和中间代码生成的主流技术，一般来说语义分析和中间代码生成是作为编译中的一“遍”同时完成的。本章首先简要介绍语义分析的主要任务，接着介绍语法制导翻译中的几个重要概念，包括文法符号的属性（综合属性和继承属性）、语法制导定义、依赖图和属性计算顺序、翻译模式等。然后详细讨论了基于 S-属性定义的自底向上属性计算方法和基于 L-属性定义的深度优先属性计算方法。这是实现语义分析和中间代码生成的两个主要方法，在第六章中将反复用到。

5.1 语义分析概述

源程序通过了词法分析和语法分析只能保证在书写上是正确的、在语法上是正确的，但不能保证其含义（语义）上的正确性。语义分析的主要任务是分析源程序的含义，并作相应的处理。语义分析模块的基本功能包括：

（1）确定类型：确定标识符所关联数据对象的类型，即处理源程序的说明部分；

（2）类型检查：对语句中运算及进行运算的运算分量进行类型检查，检查运算的合法性和运算分量类型的一致性（或者相容性），必要时作相应的类型转换；

（3）识别含义：确定程序中各组成成分组合到一起的含义，对可执行语句生成中间代码或目标代码；

（4）其它静态语义检查，如：

a）控制流检查：如对于 Pascal 语言不允许从循环外跳转到循环内、C 语言的 Break 语句引起控制离开最小包围的 while、for 等语句，需检查是否存在这样的语句；

b）唯一性检查：如标识符只能定义一次，枚举类型的元素不能重复等；

语义分析和中间代码生成是紧密联系的，在实际的编译器中语义分析和中间代码生成一般是在同一“遍”里实现的。语义分析的输入是语法分析的输出（即分析树），输出的往往是中间代码，另外它还完成了其它很多语义处理工作。

语义分析（包括中间代码生成）的主流实现技术是语法制导翻译技术，本章重点介绍编译中常用的几种语法制导翻译技术。

5.2 语法制导定义

首先为文法 G 中的每个文法符号（包括终结符号和非终结符号）引入一个**属性集合**，用以反映文法符号对应的语言结构的语义信息，如标识符的类型属性、常量的值属性、变量的地址属性等。

可以根据在语义分析过程中属性值的计算方法来对属性进行分类，属性主要有两种类型：

（1）**综合属性**（synthesized attribute）：其属性值是分析树中该结点的子结点的属性值的函数，或者说该文法符号（产生式左部非终结符号）的属性值是依赖产生式右部文法符号的属性值计算出来的；

（2）**继承属性**（inherited attribute）：其属性值是分析树中该结点的父结点和/或兄弟结点的属性值的函数，或者说该文法符号（产生式右部某个文法符号）的属性值是依赖产生式左部非终结符号及产生式右部其它文法符号的属性值计算出来的。

在语法分析过程中，如果用到了一条形如 $A \rightarrow X_1\ X_2\ \cdots X_n$ 的产生式，则分析树中必然有如图 5-1 所示的结构，其中 A 是父结点，X_1、X_2、 …、X_n 是它的子结点。

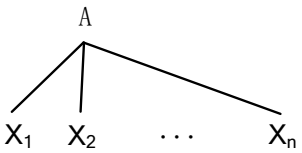

图 5-1 属性的类型

令 A 有一个属性，属性值由以下公式计算：

$$S(A) := f(I(X_1), \cdots, I(X_n))$$

其中 $I(X_i)$ 为结点 X_i 的属性，f 是一个函数，则 A 的这个属性为综合属性。

令 X_j 有一个属性，属性值由以下公式计算：

$$T(X_j) := f(I(A), I(X_1), ..., I(X_n))$$

其中 $I(A)$ 为结点 A 的属性，$I(X_i)$ 为结点 X_i 的属性，f 是一个函数，则 X_j 的这个属性为继承属性。

根据综合属性的计算方法可知，要计算分析树中父结点的综合属性，需首先计算出其子结点的属性值，因此综合属性用于“自下而上”传递属性信息。而对于某结点的继承属性，只有在已知其父结点属性值及其它兄弟结点的属性值的基础上才能够被计算，因此继承属性用于“自上而下”传递属性信息。

对于文法 G，其非终结符号（开始符号除外）既可以有综合属性也可以有继承属性，文法的开始符号 S 只有综合属性而没有继承属性。我们通常把终结符号的属性看作是综合属性，其属性值一般由词法分析器提供。

要计算文法符号的属性，需要有进行属性计算的规则。为文法的每一条产生式引入若干条语义规则，这样一种书写形式称为**语法制导定义**（Syntax-Directed Definition, SDD）。

令产生式为 $A \rightarrow X_1 X_2 \cdots X_n$，语义规则的一般形式为：

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

其中 b、c_1、c_2、…、c_k 是该产生式中文法符号的属性，f 是计算属性值的函数。若 b 是 A 的综合属性，则 $c_1, c_2, \cdots, c_k$ 是产生式右部文法符号的属性，若 b 是产生式右部某文法符号的继承属性，则 $c_1, c_2, \cdots, c_k$ 是产生式中左部或者右部文法符号的属性。

需要注意的是，在实践中综合属性和继承属性都可以依赖其自身的属性值来计算。

【例 5-1】：考察表 5-1 中的语法制导定义。

表 5-1 语法制导定义示例（只有综合属性）

产生式	语义规则
L → E n	print(E.val)
E → E_1 + T	E.val = E_1.val + T.val
E → T	E.val = T.val
T → T_1 * F	T.val = T_1.val * F.val
T → F	T.val = F.val
F → (E)	F.val = E.val
F → digit	F.val = digit .lexval

这个语法制导定义基于算术表达式文法，它对一个以 n 作为结尾标志的表达式求值，并将表达式的值打印出来。一些非终结符号带有下标，是为了区分同一个非终结符号在一条产生式里的多次出现。在这个语法制导定义中，每个非终结符号具有唯一的被称为 val 的综合属性，终结符号 digit 具有一个名为 lexval 综合属性，它的值是由词法分析器返回的整数值。

第 1 条产生式对应 1 条语义规则，功能是打印 E.val，E.val 代表整个表达式的值。

第 2 条产生式对应 1 条语义规则，功能是将产生式右边非终结符号 E_1 和 T 的 val 属性值相加，赋给产生式左边非终结符号 E 的 E.val。

第 3 条产生式对应 1 条语义规则，功能是将产生式右边非终结符号 T 的 val 属性值赋给产生式左边非终结符号 E 的 E.val。

第 4 条产生式对应 1 条语义规则，功能是将产生式右边非终结符号 T_1 和 F 的 val 属性值相乘，赋给产生式左边非终结符号 T 的 T.val。

第 5 条产生式对应 1 条语义规则，功能是将产生式右边非终结符号 F 的 val 属性值赋给产生式左边非终结符号 T 的 T.val。

第 6 条产生式对应 1 条语义规则，功能是将产生式右边非终结符号 E 的 val 属性值赋给产生式左边非终结符号 F 的 F.val。

第 7 条产生式对应 1 条语义规则，功能是将产生式右边终结符号 digit 的词法值 lexval 赋给产生式左边非终结符号 F 的 F.val。

第 1 条产生式的语义规则“print(E.val)”比较特殊，它是一个语义动作（过程或语义子程序）。对于这类语义动作，我们也把它看作是计算了一个属性，并把该属性称为（虚）综合属性。这样表 5-1 给出的就是一个只包含综合属性计算的语法制导定义。

语义规则可以用来计算属性值，也可以通过语义动作执行一些语义操作，如在符号表中登录信息、输出错误信息、进行类型检查、产生中间代码等。

【例 5-2】：考察表 5-2 中的语法制导定义。

表 5-2 语法制导定义示例（带有继承属性）

产生式	语义规则
D → T L	L.in = T.type
T → int	T.type = integer
T → real	T.type = real
L → L_1 , id	L_1.in = L.in addtype(id.entry, L.in)
L → id	addtype(id.entry, L.in)

这个语法制导定义基于一个产生变量说明语句的文法，它的作用是在变量表中登录变量的类型信息。T 的 type 属性是综合属性，用来记录变量的类型信息。L 的 in 属性是继承属性，用来传递变量的类型信息。id 的 entry 属性指向变量表中 id 的表项的入口。语义动作 addtype 执行在变量表中登录变量 id 的类型信息的动作。这个语法制导定义既包含综合属性的计算也包含继承属性的计算。

下面介绍语法制导翻译的概念。

根据语法分析中产生式对应的语义规则进行翻译的方法称为**语法制导翻译**。所谓的“语法制导”指翻译过程是基于语法分析中输出的产生式序列。这里的“翻译”是广义的，指完成语义分析的各项功能，不仅指生成中间代码。翻译是通过执行语义规则进行属性计算的方式来完成的。

从属性计算的角度来看，所谓语义分析就是要计算出分析树中每一个结点的每一个属性的值。按什么顺序来执行语义规则，即如何安排属性的计算顺序非常重要。

计算属性的语义规则的一般形式是 b := f (c_1,c_2,…,c_k)，只有在已知 c_1,c_2,…,c_k 的值的基础上，才能计算属性值 b，这种情况下我们称属性 b **依赖于**属性 c_1,c_2,…,c_k。因此在安排属性计算顺序的时候，应该先计算 c_1,c_2,…,c_k 的值，再计算 b 的值。

分析树中可能有很多结点，每个结点又可能有若干个属性，这些属性之间的依赖关系可能非常复杂。我们可以采用一个称为**依赖图**（dependency graph）的有向图来描述分析树中的结点的属性之间的依赖关系。

假设 a、b 是两个属性，先分别为 a、b 构建依赖图中的结点，若属性 b 依赖于属性 a，则从 a 结点到 b 结点画一条有向边。以表 5-1 定义的语法制导定义为例，如果语法分析时用到了产生式 E→E_1+T，则分析树中必然存在以 E 为父结点，E_1 和 T 为其子结点的结构，见图 5-2 中的虚线部分。这条产生式对应的语义规则是 E.val = E_1.val + T.val，于是为 E.val、E_1.val、T.val 分别构建属性结点，并分别从 E_1.val、T.val 的对应结点到 E.val 的对应结点画有向边，见图 5-2 的实线部分。

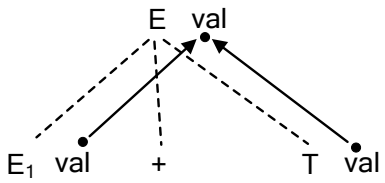

图 5-2 依赖图示例

为一棵分析树构造依赖图的算法见算法 5-1。

算法 5-1：依赖图构建算法。

输入：一个语法制导定义及一棵分析树；

输出：分析树的依赖图；

方法：执行以下算法：

```
for 分析树中的每个结点 n  do
    for 与结点 n 对应的文法符号的每个属性 a  do
        在依赖图中为 a 构造一个结点;
for 分析树的每个结点 n  do
    for 结点 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
            从结点 ci 到结点 b 构造一条有向边;
```

图 5-3 依赖图构建算法

【例 5-3】：基于表 5-1 给出的语法制导定义，给出表达式 3*5+4n 的依赖图。

表达式 3*5+4 经过词法分析后转化为单词序列 digit*digit+digit，3 个 digit 的词法值分别为 3、5、4。依赖图如图 5-4 所示，虚线部分是分析树，实线部分是依赖图。其中根结点 L 的属性结点代表 L 的虚属性（由语义动作 print 计算）。

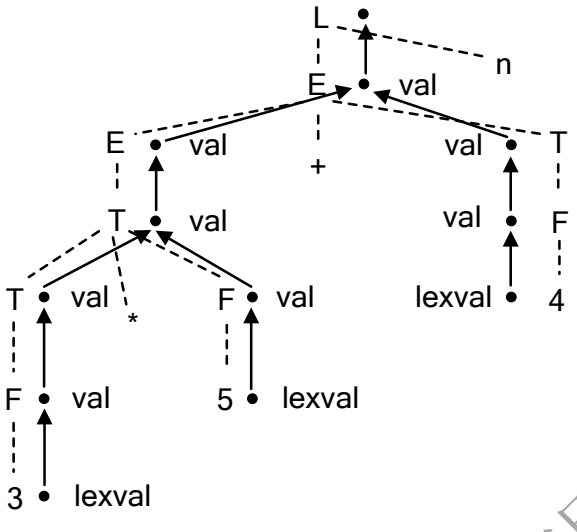

图 5-4 表 5-1 依赖图示例

表 5-1 的语法制导定义只包含综合属性的计算，综合属性用于“自下而上”传递属性值，图 5-4 有向边的走向清楚地证明了这一特性。

【例 5-4】：基于表 5-2 给出的语法制导定义，给出句子 real id_1, id_2, id_3 的依赖图。

依赖图如图 5-5 所示。其中 6、8、10 分别是 3 个 L 的虚属性（由语义动作 addtype 计算）。

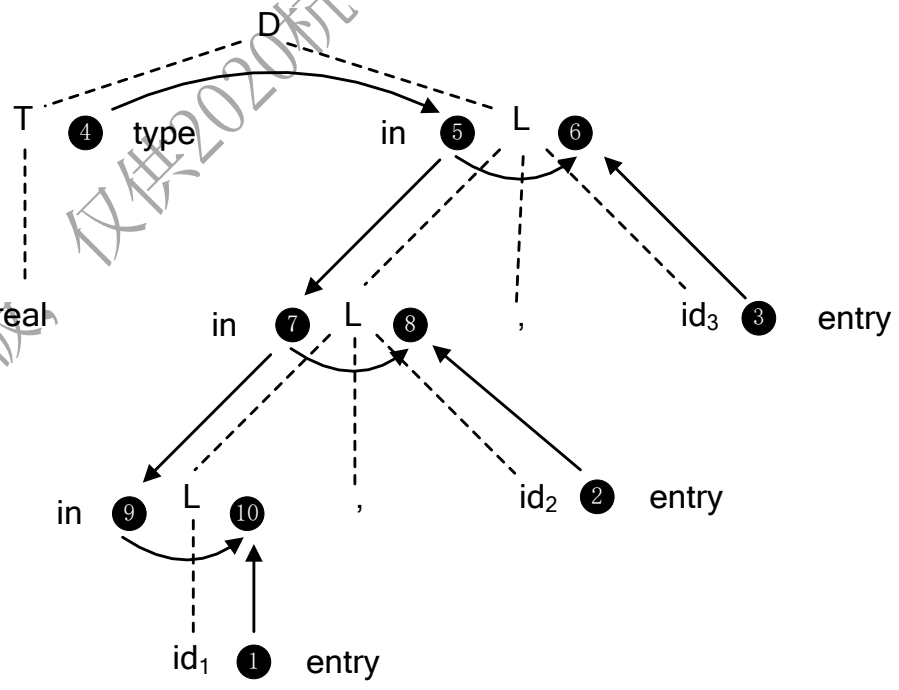

图 5-5 表 5-2 依赖图示例

依赖图刻画了对一棵分析树中不同结点上的属性求值时可能采取的顺序。如果依赖图中有一条从结点 a 到结点 b 的边，那么要先对 a 对应的属性求值，再对 b 对应的属性求值。因此所有可行的属性计算顺序应该是满足下列条件的属性结点序列 n_1、n_2、...、n_k：如果有一条从结点 n_i 到 n_j 的有向边，那么 i 必须小于 j，即 n_i 必须排在 n_j 前面。这其实就是有向无环图的**拓扑排序**（topological sort）。

如果依赖图中存在回路，那么就不存在拓扑排序，也就无法按拓扑排序的顺序安排属性的计算顺序。如果依赖图中没有回路，那么至少存在一个拓扑排序。因为没有回路，所以一定能够找到一个没有入边的结点。如果没有这样的结点，那么可以从任一结点出发不断地从一个前驱结点到达另一个前驱结点，直到回到某个已经访问过的结点，从而形成一个回路。令这个没有入边的结点为拓扑排序的第一个结点，从依赖图中删除这个点，并对其余的结点重复上面的过程，最终就可以得到一个拓扑排序。

一个依赖图的拓扑排序可能有多个，依赖图的任一拓扑排序都是一个可行的属性计算顺序。1、2、3、4、5、6、7、8、9、10 是图 5-5 中依赖图的一个拓扑排序，也是计算该依赖图中每个属性值的一个可行计算顺序。

确定语义规则的执行顺序，实现语法制导翻译主要有 3 种方法：

（1）分析树法：首先按基础文法对句子（程序）进行语法分析，构造句子的分析树；然后为分析树构造依赖图，并对依赖图作拓扑排序，得到语义规则的执行顺序；最后按顺序执行语义规则，完成属性的计算，得到句子的翻译结果。这种方法需要构建依赖图，时空效率不高，如果依赖图存在回路，这种方法无效。实践中采用这种方法的比较少。

（2）基于语义规则的方法：这种方法也是先构造分析树，然后按预先定义的策略遍历分析树来计算每个结点的属性值。语义规则的定义和翻译模式的设计（用于确定计算顺序）在编译器构造之前确定。由于分析树遍历策略的确定要考虑语义规则的定义及计算顺序，因此是基于规则的方法。这种方法的优点是不构造依赖图，不对依赖图进行拓扑排序，时空效率比较高。5.4 节介绍的 L-属性定义及其深度优先的属性计算就是这类方法。

（3）忽略语义规则的方法：这种方法将语法分析和语义分析放在同一“遍”里来实现，在进行语法分析的同时进行翻译，即边作语法分析边计算属性，计算顺序由语法分析方法确定而与语义规则无关，因此是忽略语义规则的方法。这种方法同样不构造依赖图，无需对依赖图进行拓扑排序，计算效率比较高，但是它能实现的语法制导定义比较少。接下来 5.3 节介绍的 S-属性定义及其自底向上的属性计算就是这种方法。

5.3 S-属性定义及其自底向上的属性计算

对于给定的一个语法制导定义，很难判断是否存在一棵其依赖图包含回路的分析树。如果依赖图存在回路，属性计算将变得不可能。在实践中，往往是去实现某些特定类型的语法制导定义，这类语法制导定义不会产生带回路的依赖图，S-属性定义就是这样一种语法制导定义。

只含有综合属性的语法制导定义称为 **S-属性定义**。表 5-1 就是 S-属性定义。

如果语法制导定义只含有综合属性，那么在其分析树的依赖图中有向边的走向都是“自下而上”的，参见图 5-4。这与自底向上语法分析时建立分析树的顺序是一致的。因此对于 S-属性定义可以在对句子进行自底向上语法分析的同时执行语义规则来计算各结点的属性。

具体实现方案是扩充 LR 分析器，为栈中的每一个文法符号增加一个属性域，存放分析过程中该文法符号的（综合）属性值。当用产生式进行归约时，产生式左边非终结符号入栈，其属性值由栈中正在归约的产生式右边文法符号的属性值计算。

假设在分析的某个时刻，栈顶出现了句柄“XYZ”，X、Y、Z 的综合属性存放在各自的属性域里，分别为 X.x、Y.y、Z.z。下一步操作是用产生式 A→XYZ 进行归约，在归约的同时，利用 X.x、Y.y、Z.z 计算出 A 的综合属性 A.a 并将它存放到 A 的属性域中。分析过程见图 5-6。

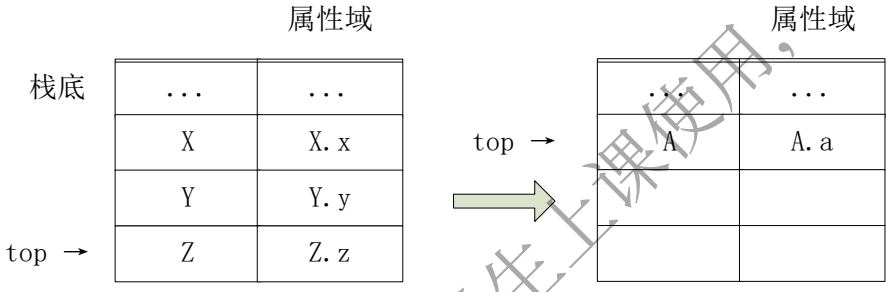

图 5-6 S-属性定义的属性计算

【例 5-5】：基于表 5-1 给出的 S-属性定义，对句子 3*5+4n 进行翻译。

句子 3*5+4n 的分析树如图 5-7 所示。

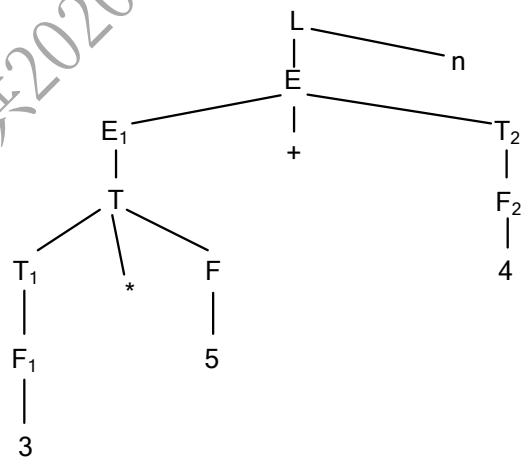

图 5-7 3*5+4n 的分析树

自底向上构建分析树，同时计算属性值。属性计算过程如下：

（1）第 1 步归约：“digit（3）归约为 F_1”，归约产生式为“F→digit”，语义规则为“F.val = digit .lexval”，计算出 F_1.val=3；

（2）第 2 步归约：“F_1 归约为 T_1”，归约产生式为“T→F”，语义规则为“T.val = F.val”，计算出 T_1.val=F_1.val=3；

（3）第 3 步归约：“digit（5）归约为 F”，归约产生式为“F→digit”，语义规则为“F.val = digit .lexval”，计算出 F.val=5；

（4）第 4 步归约：“T_1*F 归约为 T”，归约产生式为“T→T_1 * F”，语义规则为“T.val = T_1.val * F.val”，计算出 T.val=3*5=15；

（5）第 5 步归约：“T 归约为 E_1”，归约产生式为“E→T”，语义规则为“E.val = T.val”，计算出 E_1.val=T.val=15；

（6）第 6 步归约：“digit（4）归约为 F_2”，归约产生式为“F→digit”，语义规则为“F.val = digit .lexval”，计算出 F_2.val=4；

（7）第 7 步归约：“F_2 归约为 T_2”，归约产生式为“T→F”，语义规则为“T.val = F.val”，计算出 T_2.val=F_2.val=4；

（8）第 8 步归约：“E_1+T_2 归约为 E”，归约产生式为“E→E_1+T”，语义规则为“E.val = E_1.val + T.val”，计算出 E.val=15+4=19；

（9）第 9 步归约：“En 归约为 L”，归约产生式为“L→En”，语义规则为“print(E.val)”，打印出表达式的值 19。

5.4 L-属性定义及其深度优先的属性计算

实践中经常实现的另一类语法制导定义是 **L-属性定义**（L-attribute definition）。这类语法制导定义中，产生式右部各文法符号属性之间的依赖关系总是从左到右的，因此称为 L-属性的。

在 L-属性定义中，属性可以是：

（1）综合属性，因此 L-属性定义包含 S-属性定义；

（2）继承属性，但是继承属性需要满足一定的条件。假设存在一条产生式 A→$X_1X_2...X_n$，产生式右部的文法符号 X_i 有一个继承属性 X_i.a，则 X_i.a 只能依赖于：

a）产生式左部非终结符号 A 的继承属性。如果 X_i.a 依赖 A 的综合属性，依赖图中就有可能存在回路；

b）位于 X_i 左边的文法符号 X_1、X_2、...、X_{i-1} 的继承属性或者综合属性；

c）X_i 自身的继承属性或综合属性，但要求由 X_i 的全部属性组成的依赖图中不存在回路。

从分析树的角度看，计算每个继承属性的信息或者来自上边（父结点的继承属性），或者来自左边（兄弟结点的属性），或者来自自身。在实践中综合属性和继承属性都是可以依赖其自身的属性值来计算的。

【例 5-6】：判断表 5-3 是否是 L-属性定义的。

表 5-3 语法制导定义示例（非 L-属性定义）

产生式	语义规则
A → L M	L.i := l(A.i) M.i := m(L.s) A.s := f(M.s)
A → Q R	R.i := r(A.i) Q.i := q(R.s) A.s := f(Q.s)

文法符号 Q 的属性 Q.i 由其兄弟结点 R 的属性计算，因此是继承属性，但是 R 位于 Q 的右边，故该语法制导定义不是 L-属性定义的。

给定一个 L-属性定义，如何安排属性计算的顺序相对比较复杂，首先需要对语法制导定义进行改写。将语义规则放到一对花括号“{、}” 中，并插入到产生式右部的适当位置，以反映语义规则的执行顺序，这样一种书写形式称为**翻译模式**。翻译模式与语法制导定义的区别在于翻译模式中指明了语义规则的执行顺序，翻译模式是语法制导定义的一种改写。

【例 5-7】：（5.1）是一个简单的翻译模式，两个语义动作分别用花括号括起来，插入到了产生式的右部。这个翻译模式可以将表达式的中缀表示翻译成后缀表示。

E →T R

R → addop T {print (addop.lexeme)} R_1 | ε □ （5.1）

T → num {print (num.val)}

试用此翻译模式去翻译一个句子：9-5+2。

9-5+2 是一个算术表达式的中缀表示，比较常见，对我们人来说比较友好。而后缀表达式（postfix notation）是一种将运算符置于运算分量之后的表示方法。

一个表达式 E 的后缀表示可以按以下的方式递归定义：

（1）如果 E 是一个变量或常量，则 E 的后缀表示是 E 本身；

（2）如果 E 是一个形如 E_1 op E_2 的表达式，其中 op 是一个二目运算符，那么 E 的后缀表示是 $E'_1E'_2$ op，其中 E'_1 和 E'_2 分别是 E_1 和 E_2 的后缀表示；

（3）如果 E 是一个形如（E_1）的被括号括起来的表达式，则 E 的后缀表示就是 E_1 的后缀表示。

后缀表示中不需要括号，运算符的位置和它的运算分量的个数使得后缀表达式只有一种解码方式。后缀表达式的计算借助一个栈很容易实现：从左到右扫描后缀表达式，并将扫描到的符号压入栈中，直到发现一个运算符为止，然后向左从栈中找出适当数目的运算分量进行计算，并将计算结果压入栈中。重复以上操作，直到扫描完成整个后缀表达式。

根据后缀表示的规则，9-5+2 的后缀表示是 95-2+，而 9-(5+2)的后缀表达式是 952+-。翻译模式（5.1）可以将简单的中缀表达式翻译成后缀表达式。

首先构建 9-5+2 的分析树，构建过程中将用花括号括起来的语义动作作为终结符号插入到分析树中，花括号括起来的语义动作称为分析树中的**语义结点**。带语义结点的分析树如图 5-8 所示。

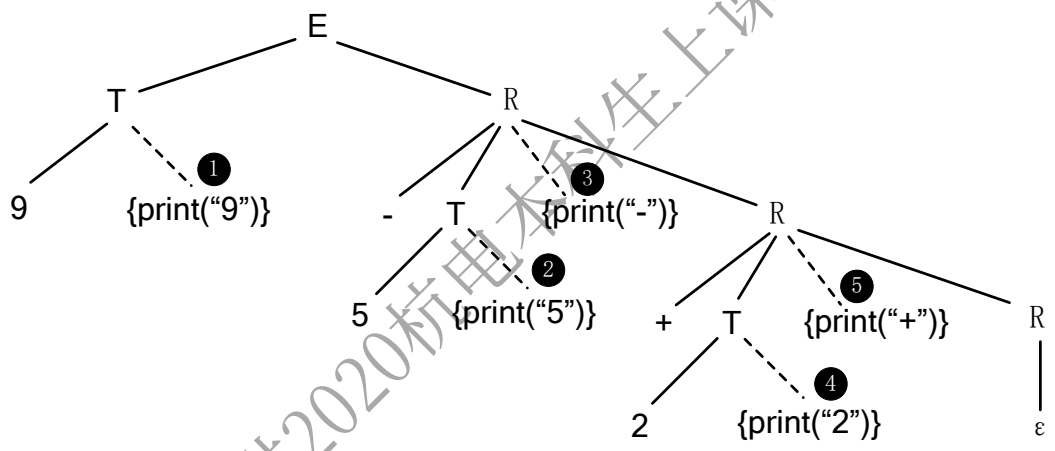

图 5-8 带语义结点的分析树

对分析树做深度优先的遍历（我们只关注语义结点的顺序），得到 5 个语义结点的排序（如图 5-8 所示），这也就是语义规则的执行顺序，按此顺序执行语义规则，可打印出表达式的后缀表示：95-2+。

通过以上例子可知翻译模式（5.1）是一个适合以深度优先顺序计算属性的翻译模式。图 5-9 是深度优先遍历示意图。这里的深度优先遍历又称为前序遍历。

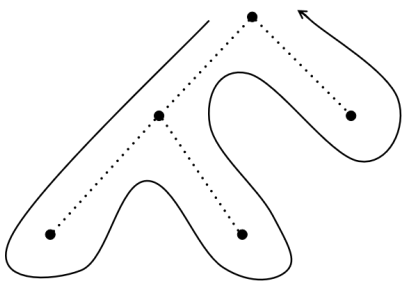

图 5-9 分析树的深度优先遍历

将一个 L-属性定义转换为一个适合以深度优先顺序计算属性的翻译模式的方法如下：

（1）把计算产生式右部某个非终结符号 A 的继承属性的语义规则插入到 A 的本次出现之前。如果 A 的多个继承属性以无回路的方式相互依赖，需要对这些属性的求值动作进行排序，以便先计算需要的属性。

（2）将计算产生式左部非终结符号的综合属性的语义规则插入到产生式右部的最右端。

【例 5-8】：表 5-4 是一个 L-属性定义，试构造其适合以深度优先顺序计算属性的翻译模式，并翻译句子(a, (a, a))。

表 5-4 L-属性定义示例

产生式	语义规则
S’ → S	S.depth = 0
S → (L)	L.depth = S.depth + 1
S → a	print (S.depth)
L → L_1, S	L_1.depth = L.depth S.depth = L.depth
L → S	S.depth = L.depth

翻译模式如下：

S’ → { S.depth = 0 } S

S → ({ L.depth = S.depth + 1 } L)

S → a { print (S.depth) }

L → { L_1.depth = L.depth } L_1, { S.depth = L.depth } S

L → { S.depth = L.depth } S

要翻译句子(a, (a, a))，首先构造句子的带语义结点的分析树，如图 5-10 所示。分析树中共有 12 个语义结点，按深度优先顺序执行语义结点中的语义规则可完成句子的翻译：

（1）S.depth = 0；

（2）L.depth = S.depth + 1 = 1；

（3）L_1.depth = L.depth = 1；

（4）S_2.depth = L_1.depth = 1；

（5）print (S_2.depth)，打印出“1”；

（6）S_1.depth = L.depth = 1；

（7）L_2.depth = S_1.depth + 1 = 2；

（8）L_3.depth = L_2.depth = 2；

（9）S_4.depth = L_3.depth = 2；

（10）print (S_4.depth)，打印出“2”；

（11）S_3.depth = L_2.depth = 2；

（12）print (S3.depth)，打印出“2”。

该句子的翻译结果是打印出：122。通过分析可知，该翻译模式的功能是输出句子中每个 a 的嵌套深度。

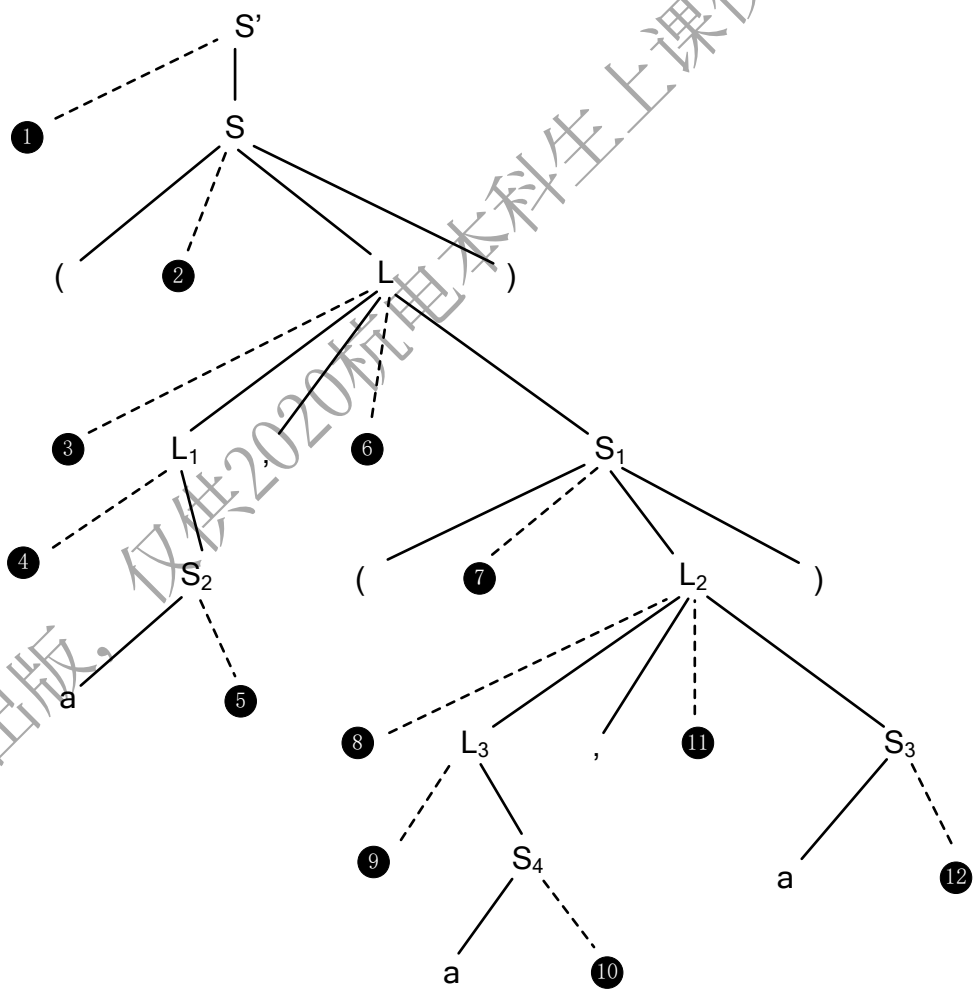

图 5-10 句子(a, (a, a))的带语义结点的分析树

5.5 小结

语法制导翻译技术是一种用途广泛的信息处理技术，其应用不仅限于语言编译领域。本章介绍语法制导翻译技术是为编译过程中的语义分析和中间代码生成提供解决方案。高级语言文法中的文法符号代表语法单位，每个语法单位都有“语义”，可以通过为每个文法符号引入一个属性的集合来描述它们的“语义”。属性按照计算方法的不同分为综合属性和继承属性，综合属性用于“自底向上”传递属性值，继承属性用于“自顶向下”传递属性值。属性可以通过执行语义规则来计算，为文法产生式引入一套计算属性的语义规则可以将文法扩展为语法制导定义。要对一个句子（程序）进行语义分析，就是要计算出该句子的分析树中每个结点的每个属性值。无论采用哪种语法分析方法来构造句子的分析树，都需要用到一个产生式的序列，执行这个产生式序列对应的语义规则来计算分析树中结点的属性值就是语法制导翻译。属性计算的顺序需要遵循属性之间的依赖关系，分析树中所有结点的属性之间的依赖关系可以用一个依赖图来描述，这个依赖图不能存在回路，否则无法完成所有翻译。基于 S-属性定义的自底向上属性计算方法和基于 L-属性定义的深度优先属性计算方法是语义分析中常用的两种方法。S-属性定义只包含综合属性，而综合属性的传递方向是“自底向上”的，因此可以在自底向上语法分析的同时实现句子的翻译。L-属性定义可以包含继承属性，但是这个继承属性只能依赖它产生式中左边文法符号的属性。基于 L-属性定义来翻译句子，首先需要将 L-属性定义改写为一个适合以深度优先顺序计算属性值的翻译模式。本章的重点是理解属性计算的几个相关概念，包括属性的定义、语义规则与语法制导定义、依赖图与属性计算顺序、翻译模式等；掌握 S-属性定义和 L-属性定义的特点及适用范围，掌握基于这两类语法制导定义对句子进行翻译的方法。

习题

5.1 已知程序的文法 G[P]如下：

$$P \rightarrow D$$

$$D \rightarrow D; D \mid id: T \mid proc\ id; D; S$$

(1) 请写一个语法制导定义，输出程序中一共声明了多少个 id。

(2) 请写一个翻译模式，输出程序中每个变量 id 的嵌套深度。

提示：使用一个综合属性 c 表示程序中声明的 id 个数，使用一个继承属性 d 表示嵌套深度。

5.2 用 S 的综合属性 val 给出下面文法 G[S]中 S 产生的二进制数的值。（如输入 101.101 时，S.val = 5.625）

S → L . L | L

L → L B | B

B → 0 | 1

提示：二进制转换为十进制分为整数部分的转换和小数部分的转换，转换规则如下例：

1101.101 = 1*2^3+1*2^2+0*2^1+1*2^0+1*2^(-1)+0*2^(-2)+1*2^(-3)

= 8+4+0+1+0.5+0+0.125 = 13.625

因此，可以给 B、L、S 设置综合属性 val，表示该符号代表的十进制值，给 L 另外设置一个表示其长度的综合属性 length。

5.3 设有文法 G[E]：

E → E + T | T

T → num.num | num

该文法产生仅有+运算的算术表达式，运算对象可以是整数和实数，文法中 num 代表数字串。

(1) 试给出计算表达式结果类型的语法制导定义。要求当两个整型数相加时，结果仍为整型，否则结果为实型。

(2) 扩充上面的语法制导定义，使之能把表达式翻译成后缀形式，同时也能确定结果的类型。注意使用一元运算符 inttoreal 把整型数转换为实型数，int+和 real+分别表示整型数加法运算和实型数加法运算。

5.4 给定文法 G[A]的如下翻译模式：

A→aB { print “0”}

A→c { print “1”}

B→Ab { print “2” }

假设在按某一个产生式进行归约时，将立即执行相应的语义动作，试问，当输入为 aacbb 时，打印出的字符串是什么？画出注释分析树，并给出相应的分析过程。

5.5 给定表达式文法 G[S’]，其中一个表达式的“值”通过以下语法制导翻译来描述：

产生式	语义动作
(1)S’→E	{print(E.val)}
(2) E→E_1 + E_2	{E.val = E_1.val + E_2.val}
(3) E →E_1*E_2	{E.val = E_1.val * E_2.val}
(4) E →(E_1)	{E.val = E_1.val}
(5) E→n	{ E.val =n.lexval}

如果采用 LR 分析技术，给出表达式(5*4+8)*2 的语法树，并在各结点注明属性 val 的具体值。

5.6 假设 G[D]是某个语言用于生成变量类型说明的文法：

D → id L

L → ,id L | ; T

T → integer | real

构造一个翻译方案，仅使用综合属性，把每个标识符的类型填入符号表中。