第五章 语法制导翻译

语法分析
语义分析
中间代码生成 } 语义翻译 } 语法制导翻译 / CFG

语法制导翻译使用CFG来引导对语言的翻译，是一种[面向文法]的翻译技术

· 如何表示语义信息：为CFG中的[文法符号]设置[语义属性]，用来表示语法成分对应的[语义信息]

· 如何计算语义属性：文法符号的语义属性求值是用文法符号所在产生式(语法规则)相关联的[语义规则]来计算的

· 语法制导定义(SDD)：是对CFG的推广：①将每个[文法符号]和一个[语义属性]集合相关联 ②将每个[产生式]和一组[语义规则]相关联

· 语法制导翻译方案(SDT)：SDT是在产生式右部嵌入了程序片段的CFG，这些程序片段称为[语义动作]。按照惯例，语义动作放在花括号内。

5.1 语法制导定义 SDD (定义同上)

· 文法符号的属性：综合属性、继承属性

非终结符：子结点/本身　父结点/兄弟结点/本身

终结符：从词法分析器处获得[综合属性值]

· 注释分析树：每个节点都带有属性值的分析树

· 属性文法：一个没有副作用的SDD有时也称为属性文法

属性文法的规则仅仅通过其它属性值和常量来定义一个属性值

· SDD的求值顺序：依据属性之间的依赖关系

· 依赖图：描述了分析树中结点属性间依赖关系的有向图

如果图中没有环，那么至少存在一个拓扑排序

5.2 S-属性定义与L-属性定义

· 仅仅使用[综合属性]的SDD称为[S属性的SDD]或[S-属性定义]、S-SDD

S-属性定义可以在自底向上的语法分析过程中实现

· L-属性定义(L属性的SDD或L-SDD)：依赖图的不能从右到左

· L-SDD的正式定义：一个SDD是L-SDD，当且仅当它的每个属性要么是一个综合属性，要么满足如下条件的继承属性：假设存在一个产生式 $A \to X_1 X_2 \cdots X_n$，其右部符号 $X_i (1 \le i \le n)$ 的继承属性仅依赖于下列属性：①A的继承属性 ②产生式中 X_i 左边的符号 $X_1, X_2, \cdots X_{i-1}$ 的属性。③ X_i 本身的属性，但 X_i 的全部属性不能在依赖图中形成环路。

· 虚属性：eg. print(...)，addtype(...)

5.3 语法制导翻译方案 SDT

· 语法制导翻译方案(SDT)是在产生式右部中嵌入了程序片段(语义动作)的CFG

· SDT可以看作是SDD的具体实施方案

怎么算 / 什么时算　　\ 怎么算

①将S-SDD转换为SDT

· 将一个S-SDD转换为SDT的方法：将每个语义动作都放在产生式的最后

· 如果一个S-SDD的基本文法可以使用LR分析技术，那么它的SDT可实现

· 语法分析器的扩展：为每个栈记录增加[属性值字段]，存放文法符号的[综合属性值]。在每次[归约]时[调用]计算综合属性值的[语义子程序]。

· 若支持多个属性：法1：使栈记录变得足够大。法2：在栈记录中保存指针。

②将L-SDD转换为SDT

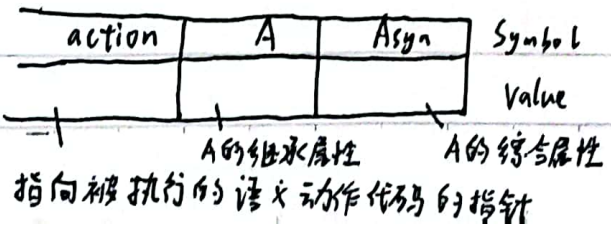
· 将计算一个产生式主部的符号的[综合属性]的动作放置在这个产生式右部的[最右端]；将计算某个非终结符号A的[继承属性]的动作插入到产生式右部中[紧靠在A的本次出现之前]的位置上。

· 如果一个L-SDD的基本文法可以使用LL分析技术，那么它的SDT可以在LL或LR语法分析过程中实现。

5.4 L-SDD的自顶向下翻译　(在预测分析的同时实现语义翻译)

5.4.1 在非递归的预测分析过程中进行翻译

扩展语法分析栈



· 分析栈中的每一个记录都对应着一段执行代码

综合记录[出栈]，要将[综合属性值]复制给后面特定语义动作

变量展开时（即变量本身[出栈]时）如果其含有继承属性，则要将[继承属性值]复制给后面的特定的语义动作.

5.4.2 在递归的预测分析过程中进行翻译

算法：· 为每个[非终结符A]构造一个[函数]，A的每个[继承属性]对应函数的一个[形参]，函数的[返回值]是A的[综合属性值]。对出现在A产生式中的每个文法符号的每个[属性]都设置一个[局部变量]

· [非终结符A]的代码根据当前的输入决定使用哪个产生式

5.5 L-属性定义的自底向上翻译

给定一个以LL文法为基础的L-属性定义，可以修改这个文法，并在LR语法分析过程中计算这个新文法上的SDD

· 首先构造SDT，在各个[非终结符之前]放置语义动作来计算它的[继承属性]，并在[产生式后端]放置语义动作计算[综合属性]

· 对于每个内嵌[语义动作]，向文法中引入一个[标记非终结符]来替换它。每个这样的位置都有一个[不同的]标记，并且对于任意一个标记[M]都有一个产生式[$M \to \varepsilon$]

· 如果标记非终结符M在某个产生式中$A \to \alpha\{a\}\beta$中替换了语义动作a，对a进行修改得到a'，并且将a'关联到$M \to \varepsilon$上。动作a'复制

(a) 将动作a需要的A或α中符号的任何属性作为M的[继承属性]进行

(b) 按照a中方法计算各个属性，但是将计算得到这些属性作为M的[综合属性]

第六章 中间代码生成

6.1 声明语句的翻译

- 声明语句翻译的主要任务：收集标识符的[类型]等属性信息，并为每一个名字分配一个[相对地址]

名字的[类型]和[相对地址]信息保存在相应的[符号表]记录中

- 基本类型是类型表达式
- 可以为类型表达式表命名，类型名也是类型表达式
- 将[类型构造符]作用于[类型表达式]可以构成新的类型表达式

  数组构造符 array 指针构造符 pointer 笛卡尔乘积构造符 X

  函数构造符 -> 记录构造符 record

- 局部变量的存储分配 enter(name, type, offset)

从[类型表达式]可以知道该类型在[运行时刻]所需的存储单元数量称为[类型的宽度](width)

在[编译时刻]，可以使用[类型的宽度]为每一个名字分配一个[相对地址]

6.2 赋值语句的翻译

6.2.1 简单赋值语句的翻译

- 赋值语句的基本文法 详见 PPT P18
- 赋值语句翻译的主要任务：生成对表达式求值的三地址码

newtemp(): 生成一个新的临时变量 t，返回 t 的地址

gen(code): 生成三地址指令 code

lookup(name): 查询符号表，返回 name 对应的记录

- 增量翻译 (Incremental Translation)：在增量方法中，gen() 不仅要构造出一个新的三地址指令，还要将它添加到至今为止已生成的指令序列之后

6.2.2 数组引用的翻译

- 将[数组引用]翻译成[三地址码]时要解决的主要问题是确定[数组元素的存放地址]，也就是数组元素的寻址

· 数组元素寻址　一维数组 base + i×w
基地址　偏移地址

二维数组 base + $i_1 \times w_1 + i_2 \times w_2$

k维数组 base + $i_1 \times w_1 + \cdots + i_k \times w_k$

eg. type(a) = array(3, array(5, int))

addr(a[i_1][i_2]) = base + $i_1 \cdot 20 + i_2 \cdot 4$

· 数组元素寻址SDT　L的综合属性 L.type, L.offset, L.array
eg.array(8, int)← 数组元素的类型　w_j　数组名在符号表的入口地址

· 符号表的组织——数组　基本属性
名字　种属　类型　地址　扩展属性指针　扩展属性

6.3 控制语句的翻译

· 控制流语句的基本文法　详见PPT P41

· 控制流语句的代码结构

eg. S -> if B then S_1 else S_2.

布尔表达式B被翻译成由[跳转指令]构成的跳转代码

继承属性: B.true, B.false, S.next　用指令的标号标识一条三地址指令

· newlabel(): 生成一个用于存放标号的新的临时变量L,返回变量地址

label(L): 将下一条三地址指令的标号存放到地址L中

· 控制流语句SDT编写要点

·分析每一个非终结符之前:先计算[继承属性],再观察代码结构图中该非终结符对应方框顶部是否有[导入箭头]。如果有,[调用 tabl label 函数]

· 上一个代码框执行完[不顺序]执行下一个代码框,生成一条[显式跳转指令]

· 有[自下而上的箭头]时,设置[begin属性]。且定义后直接调用[label()函数]绑定地址

· 布尔表达式的翻译　布尔表达式的基本文法 详见PPT P51

· 在跳转代码中,逻辑运算符&&、||和!被翻译成[跳转指令]。运算符本身不出现在代码中,布尔表达式的值是通过代码序列中位置来表示的.

· 基础文法:可以使用LR分析技术

· SDD：L-SDD

赋值语句：只定义了[综合属性]

分支、循环语句：只定义了[继承属性]，且不依赖右兄弟节点[属性值]

· SDT的通用实现方法：首先[建立]一棵语法[分析树]

然后按照[从左到右]的[深度优先]顺序来[执行这些动作]

if-then语句，$B \to E_1 relop E_2$语句，$B \to B_1 or B_2$语句，$B \to B_1 and B_2$语句的修改 详见PPT P67-70

6.4 回填

基本思路：生成一个跳转，暂时不指定该跳转指令的[目标标号]。这样的指令都被放入由跳转指令组成的[列表]中。[同一个列表中的所有跳转指令具有相同的目标标号]。等到能够确定正确的目标标号，才去填充这些指令的目标标号。

· 非终结符B的综合属性　　B.truelist，B.falselist 包含跳转指令的列表

makelist(i)：创建一个只包含i的列表，i是跳转指令的标号，函数返回指向创建的新列表的指针

merge(p_1,p_2)：将p_1和p_2指向的列表进行合并，返回指向合并后的列表的指针

backpatch(p,i)：将i作为目标标号插入到p所指列表中的各指令中

nextquad：即将生成的下一条指令的标号。

布尔表达式的回填 详见PPT P78-88

· 控制流语句的回填　　S.nextlist　综合属性

· 回填技术SDT编写要点

文法改造：在list[箭头指向的位置]设置[标记非终结符M]

在产生式末尾的语义动作中：计算[综合属性]，调用backpath()函数[回填]各个list，生成必要的附加指令。

6.5 swich语句的翻译

· swich语句的两种翻译 详见PPT P100-101

· 在代码生成阶段，根据分支的个数以及这些值是否在一个[较小的范围内]，这种条件跳转指令序列可以被翻译成[最高效的n路分支]

· 指令 case Vi Li 和 if t=Vi goto Li 的含义相同，但是 case 指令更加容易被最终的代码生成器探测到，从而对这些指令进行特殊处理.

6.6 过程调用语句的翻译

· 需要一个队列 q 存放 $E_1.addr$、$E_2.addr$、……、$E_n.addr$.

S -> call id(Elist) {n=0; for q 中每个 t do { gen('param' t); n=n+1;} gen('call' id.addr ',' n)}

Elist -> E {将 q 初始化为只包含 E.addr}

Elist -> Elist, E {将 E.addr 添加到 q 的队尾}