《编译器设计第二版》笔记

KenshinLiu

Last Update: 2022 年 3 月 15 日

编译器检查变量在使用前已经声明

要达到的效果：要求某些类别的变量在使用前已经声明，但允许程序员将声明和可执行语句混合起来。

解决方案：编译器创建一个名字表。编译器在处理声明时向表中插入一个名字，而在每次引用名字时去表中查找，查找失败就表示缺少对应的声明。

类型系统判断类型是否等价的两种方案

名字等价：该规则断言两个类型等价的充分必要条件是二者同名，认为相同的类型名即代表同一种类型。

结构等价性：该规则断言两个类型等价的充分必要条件是二者具有相同的结构。如果两个对象由同一组字段组成，且字段排列顺序相同，对应的字段具有等价的类型，则这两个对象为同一种类型。

属性求值的方法

(1) **动态方法**这种技术使用特定的属性化语法分析树的结构，来确定求值次序。Knuth 关于属性语法的原始论文提出了一种求值程序，以类似于数据流计算机体系结构的方式运作，即每个规则在其所有操作数就绪后即“击发”。实际上，这可以使用就绪属性（即可求值的属性）的队列来实现。随着对每个属性的求值，求值程序会检查其在属性依赖关系图中的后继属性，判断后继属性“就绪”与否（参见 12.3 节）一种相关的方案是建立属性依赖关系图，对其拓扑排序，使用拓扑次序对属性进行求值。

(2) **无关方法**在这一类方法中，求值的次序与属性语法和特定的属性化语法分析树都是无关的。大体上，系统的设计者可以从其自身的考虑出发，选择一种他认为适合于属性语法和求值环境的方法。这种风格的求值方法包括：从左到右重复多趟（直至所有属性的值都确定为止）、从右到左重复多趟和从左到右与从右到左交替多趟处理。这些方法有简单的实现，其运行时开销也相对较小。当然，它们也缺之根据对特定属性语法树的认识进行改进的能力。

(3) **基于规则的方法**基于规则的方法依赖于对属性语法的静态分析，来构造出一个求值次序。在该框架下，求值程序依赖于语法结构，因而，对规则的应用受到了语法分析树的引导。在有符号进制数的示例中，对产生式 4 的求值次序应该使用第一个规则设置 Bit.position，递归向下到 Bit，返回后，使用 Bit value 设置 List.value。。类似地，对于产生式 5，它应该首先对前两个规则求值，以便为产生式的右侧定义 position 屈性，然后递归向下来处理各个子结点。在返回后，就可以对第三个规则求值，来设置父结点 List 的 List.value 字段。如果

工具能够离线执行必要的静态分析，那么可以利用这种工具来生成快速的基于规则的求值程序。

DAG

DAG 是指有向非循环图。在 DAG 中，结点可以有多个父结点，相同子树可以被重用，这种共享使得 DAG 更为紧凑。

DAG 是具有共享机制的一种 AST。

控制流图

程序中最简单的控制流单位是一个基本程序块，即（最大长度的）无分支代码序列。它开始于一个有标号的操作，结束于一个分支、跳转或条件判断操作。

控制流图 (Control-Flow Graph, CFG) 用一个结点表示每个基本程序块，用一条边表示块之间的每个可能的控制转移。

词法作用域和动态作用域

词法作用域和动态作用域之间的区别，只出现在过程引用在自身作用域之外声明的变量时，这种变量通常称为**自由变量**。

词法作用域规则：自由变量绑定到词法上与使用位置最接近的同名声明。如果编译器从包含使用处的作用域开始处理，并连续检查外层的作用域，变量将绑定到找到的第一个声明。声明总是来自包含引用处的一个作用域中。

动态作用域规则：自由变量绑定到在运行时最近创建的同名变量。因此，在执行遇到自由变量时，即将绑定到改名字最新创建的实例。

AR 与栈的关系

为实现过程调用和作用域化的命名空间这两个“孪生”抽象，编译器在转换时必须建立一组运行时结构。在控制和命名两方面都涉及的一个关键数据结构是活动记录（Activation Record, AR），这是与对特定过程的特定调用相关联的一块私有内存。

如果过程的激活期超过器调用者的激活期，那么在栈上分配 AR 的规范就被破坏了。类似地，如果过程可能返回一个对象（如闭包），其中显式或隐式引用了已返回过程的局部变量，那么在栈上分配活动记录是不合适的，因为会出现悬挂指针。在这种情况下，AR 可以保存在堆中。

面向对象语言中符号表的使用

> TODO 三种表具体指什么表，整理笔记的时候完善下

为了支持面向对象语言中类层次之类的命名环境，编译器通过特定的链接顺序组成符号表的链接集合。在面向对象语言中，编译器使用比类 Algol 语言中更多的表，且使用这些表的方式必须反映实际的命名环境。编译器可以将这些表按适当的顺序链接在一起，也可以分别维持三种表，并按适当的顺序查找表。

某些面向对象语言的主要复杂性并不在于类层次结构的存在，而是从何时起定义了该层次结构。如果面向对象语言要求类定义出现在编译时且编译后不能发生改变，那么方法内部的名字解析可以在编译时进行。我们说这样的语言有封闭的类结构（closed class structure）。在另一方面，如果语言允许运行的程序改变类结构，或者像 Java 那样在运行时导人头，或者像 Smalltalk 那样允许在运行时编辑类，那么，这样的语言有开放的类结构（ open class structure).

> **封闭的类结构** 如果应用程序的类结构在编译时就已经固定下来，那么这种面向对象语言具有封闭的类层次结构。
> **开放的类结构** 如果应用程序可以在运行时改变其类结构，那么语言具有开放的类层次结构。

面向对象中继承关系的实现

见 6.3.4 节。

过程之间值的传递

参数绑定将调用位置处的实参映射到被调用者的形参。它使得程序员编写过程实现时，无需关注调用过程的上下文信息。它还使得程序员可以从许多不同的上下文环境中调用过程，而不会对每个调用者暴露过程内部操作的细节。

大多数现代程序设计语言使用两种惯例之一将实参映射到形参：传值（call-by-value）绑定和传引用（call-by-reference）绑定。

> **传值调用**
> 一种约定，调用者对实参求值，并将其值传递给被调用者。在被调用者中对值参数的修改在调用者中是不可见的。

传值绑定的一种变体是传值兼传结果（call-by-value-result）绑定。在值-结果方案中，在控制从被调用者返回调用者时，会将形参的值反向复制到对应的实参。Ada 和 FORTRAN 77 语言包含了值-结果参数。

传引用调用

一种约定，编译器将实参的地址传递给被调用者中对应的形参。如果实参是一个变量（而非表达式），那么改变形参的值会导致实参的值同样发生改变。

确定可寻址性

作为链接约定的一部分，编译器必须确保每个过程对其需要引用的每个变量都能产生一个地址。在类 Algol 语言中，过程可以引用全局变量、局部变量和外层词法作用域中声明的任何变量。一般来说，地址计算包括两个部分：在目标值所处的作用域中，找到包含目标值的适当数据区的基地址，找到值在该数据区内部的正确偏移量。获得基地址的问题分为两部分：具有静态基地址的数据库，和直至运行时才能得知基地址的数据区。

数据区

为特定作用域保存数据的内存区称为该作用域的数据区。

基地址

数据区的开始地址通常称为基地址。

具有静态基地址的变量

在编译器通常的安排下，全局数据区和静态数据区具有静态基地址。为此类变量产生地址的策略很简单：酸楚该数据区的基地址，置于寄存器中，再加上变量相当于基地址的偏移量。

要生成静态基地址的运行时地址，编译器需要对数据区添加一个符号性的、汇编层次的标号。取决于具体的目标机指令集，该标号可以用于加载立即数的 load 操作中，也可以用于初始化一个已知的内存位置，在后一种情况下，可以用标准的 load 操作将其加载到寄存器中。

具有动态基地址的变量

过程内部声明的局部变量通常存储在过程的 AR 中。因而，它们具有动态的基地址。为访问这些值，编译器需要一种机制来找到各个 AR 的地址。幸运的是，词法作用域规则限制了在代码中任意位置所能访问的 AR 的集合，该集合只包含当前 AR 及词法山包含当前过程的其他过程的 AR。

如果访问的是当前过程的局部变量。其基地址只是当前 AR 的地址，存储在 ARP 中。因此，编译器可以输出代码，将变量对应的偏移量与 ARP 相加，并将结果作为变量的地址。

如果访问的是其他过程的局部变量的话则稍微复杂一点。为了访问某个外层词法作用域的局部变量，编译器必须设法构建一些运行时数据，将语法分析器中使用词法上作用域化的符号表产生的静态坐标，映射到运行时地址。

例如，假定词法层次 m 上的过程 fee，引用了 fee 的词法祖先 fie 中的变量 a，fie 位于层次 n。语法分析器将该引用转换为一个静态坐标 $< n, o >$，其中 o 是 a 在 fie 的 AR 中的偏移量。编译器可以计算 fee 和 fie 之间词法层次的数目，即 $m - n$。（坐标 $< m - n, o >$ 有时称为该引用的静态距离坐标）

编译器需要一种机制将 $< n, o >$ 转换为运行时地址。一般来说，该方案将使用运行时数据结构，来找到最接近的、词法层次为 n 的过程的 AR，并使用相应的 ARP 作为地址计算的基地址。将偏移量 o 与基地址相加，即可产生一个运行时地址，对应于静态坐标为 $< n, o >$ 的值。接下来将介绍两种常用的简历运行时数据结构的方案，存取链（access link）和 GD（Global Display）。

存取链

存取链的原理比较简单。编译器确保每个 AR 都包含一个指针，称为存取链或静态链，指向其在词法上紧邻的祖先。从当前过程起，各个存取链形成了一个链表，包括了当前过程在词法上的所有祖先。

GD

在这种方案中，编译器分配一个单一的全局数组，称为 display，来保存各个词法层次上最新激活的过程的 ARP。对其他过程的局部变量的所有引用，都可以通过 display 变为间接引用。

使用 GD 的情况下，非局部访问的代价是固定的。使用存取链的情况下，编译器会生成 $m - n$ 次 $load$ 操作。

标准化链接

过程的链接属性是编译器、操作系统、目标机之间的一个契约，对命名、资源分配、可寻址性和保护等职责进行了清楚的划分。过程的链接属性，确保将被编译器转换的用户代码和其他来源的代码之间的互操作性。