计算机科学2003Vol. 30№. 4

一个基于对象、域和型的层次式程序表示模型*)

赵 洋 蔡志旻 金 浩 潘金贵

(南京大学计算机软件新技术国家重点实验室 南京大学计算机科学与技术系 南京210093)

A Hierarchical Program Visual Model Based on Object, Scope and Type

ZHAO Yang CAI Zhi-Min JIN Hao PAN Jin-Gui

(State Key Laboratory for Novel Software Technology at Nanjing University, Department of Computer Science and Technology of Nanjing University, Nanjing 210093)

Abstract It is necessary to represent a program in a visual model before analyzing it. This paper introduces a hierarchical program visual model based on object, scope and type. It depicts a program's information through a recursive-nested structure. Particularly, it emphasizes on the inherent relationships among objects, scopes and types and connects objects and types within scopes. This paper explains all parts and their relationships in details and also addresses briefly about how to realize the model.

Keywords Object, Scope, Type, Template, Hierarchical model

1 引言

计算机程序是以某种程序设计语言为工具编制出来的指令序列,它表达了程序员的思想。高级程序设计语言作为一种符号语言,已经成为人机对话的工具。人们用某种或某几种高级语言写出程序来表达自己想做的事和期望达到的效果,计算机接受这些程序然后运行产生相应的结果。

对象、域和型是程序设计的基本概念。然而,由于三者关系的复杂性,开发者经常会在实践中犯错误,导致实际编写出来的程序与其希望的并不一致,可能小有偏差,也可能大相径庭。比如,类型的不匹配和域层次的无意覆盖就是最容易出现的错误。例如,对于下面的一个用C语言书写的程序片断:

```
unsigned int data;
int var;
…
data=var;
```

从语法上看是正确的,编译一般也是可以通过的。也就是说,它是一个"语法正确"的程序。但是,通过仔细观察,我们就会发现该片断中的赋值语句隐藏着缺陷:赋值语句的左侧是一个无符号型的整数,而右侧却是一个有符号型的整数。我们知道,有符号数使用其最高位来表示符号。当把一个有符号的负数赋值给一个无符号型的整数时将会丢掉符号而使该负数变成一个非常大的正整数,常常使得程序在实际运行中发生不可预见的错误;然而,这并不是开发者的本意。这就是一个类型使用不匹配的例子。

现在,检查语法正确与否由编译器来完成。但是,对于以上这种潜在的错误,编译器就无能为力了。因此,有必要在编译之前对程序进行静态的分析来检测是否存在类似的问题。

我们如何去判断使用了给定的程序设计语言编制出来的程序是不是符合语法规范?是不是符合程序员的要求呢?这里就涉及到一个对程序的判定问题。进行判定的前提是对程序进行规范化的处理,最好是建立一种特定的表示程序细节的模型,然后再对这个模型进行分析。迄今,已经出现了许多的程序表示模型,不同的模型反映的是程序不同方面的特性;更重要的是,模型的特征对于构筑于其上的高层应用有很大的影响。本文提出了一种基于对象、域和型的层次式程序表示模型(以下简称程序模型),它使用了层次式的结构来表示给定的程序,具有自身的独特性和有效性。

2 对象、域和型

首先,我们来介绍模型中涉及的对象、域和型的含义。

对象(Object)是具有类型属性和域属性的占用一定内存空间的实体,并有一个对象名来标识。对于面向对象的程序设计语言来说,对象概念的本质是把数据和作用于数据的操作封装为一个整体。它所定义的对象可以说是对客观世界实体的直接模拟,按照封装的方法构造与客观世界成分相对应的软件模块。而在本文中提出的程序模型中对象的含义与以上所说的有所不同。它不仅包括了面向对象的程序设计中的对象变量,还包括结构化程序设计中出现的简单变量、常量、枚举成员等,而且一个函数也可以看作是一个对象。显然,这里的对象是一种更加广泛意义上的概念(在本文中如没有特别说明,对象均为此含义)。

域(Scope)或作用域是使得实体的引用和实体的名称保持一致性的区域,包括了类型和对象的作用域。类型的作用域就是该类型的有效范围。对于系统定义的基本类型如int, char等显然具有全局的作用域,而对于用户定义的类型如struct, typedef等类型只具有特定的作用域。对象的作用域是对象名与对象地址保持一致性的范围。下面,我们以面向对象的程序设计中的类作用域来举例,它的域就是指类定义和相应的成员函数定义的范围。在该范围内,一个类的成员函数对于同一类的数据成员具有无限制的访问权。一个类的所有成员位于这个类的作用域中,它们都能访问同一类中的其他任

*)本课题得到日本富士通公司国际合作研究基金的资助。赵 洋 硕士研究生,研究方向为软件开发环境及多媒体信息处理技术。蔡志旻 金 浩 硕士研究生,研究方向为中间件技术,软件工程应用技术。潘金贵 教授,博士生导师,主要研究领域为中间件、Agent技术及多媒体信息处理等。

意的成员。对于类作用域外的一个类的数据成员和成员函数的访问则受到类可见性的控制，只能通过接口或继承等手段受限访问。这种作用域思想是要把一个类的数据结构和功能封装起来，从而使得在类的成员函数之外对类的数据进行访问是有限的。这里所说的类作用域在本程序模型中的名称为StructureScope。此外，还必须提到名字空间，名字空间可以把一个全局的名字空间分成多个可管理的小空间(NameSpaceScope)，也就是某一个名字在其中必须唯一的作用域。特别地，大部分程序设计语言规定：一个名字在同一个域不能同时指两种以上类型或非类型名(变量名、常量名、函数名、对象名或枚举成员)。

型(Type)表明了对象的属性。对于任何一个对象，只有确定了其类型，才能确定对象的值集和作用于对象的操作集。具体到C/C++的语法，它的型可以分为以下几个类别：①基本数据类型，例如：整型(int)、字符型(char)、布尔型(bool)等；②非基本型，如数组型(type [])、指针型(type *)等；③用户自定义型，如结构型(struct)、枚举型(enum)、类(class)等。本文中所提出的程序模型不仅仅包括以上三种数据类型，还包含有函数型(FunctionType)、类型定义型(TypedefType)等。

此外，模板(Template)是现代面向对象语言的一个重要的特性，它使得开发者能够快速建立类型安全的类库集合和函数集合。模板是一种安全的、高效的重用代码的方法，它的实现，方便了大规模的软件开发。模板不是通过继承和组合重用对象代码，而是一种基于类型的重用。它通过动态联编参数类型，使得在创建对象或调用函数时，可以通过传递类型参数来改变其运行时刻的实际行为。模板本质上仍然是类型，但是在本模型中把模板单列出来，原因是因为模板是相对较特殊的类型，具有和普通类型很多不同的特征，并且这样有助于本模型的使用者根据模板的特性更好地把握其使用。

以上所介绍的四种要素，构成了我们的层次式程序模型，如图1中灰色部分所示。

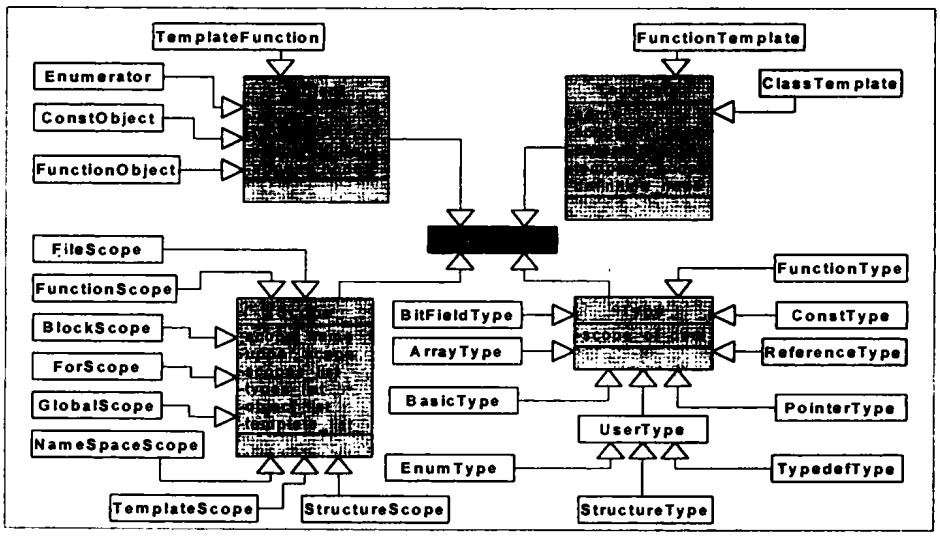

图1 程序模型的结构示意图

3 程序模型中各种类的继承关系

对任何一种程序设计语言来说，其语法规范的适应性是非常广泛的，甚至是千姿百态的，单以对象、域、型和模板四者去表示程序中所有的语法单位显然是不够的，我们对程序准确的理解和分析是建立在对程序的精确建模的基础之上的。因此有必要对上述四个部分进行逐步的细化使得精确匹配程序中的不同的语法规则，如图1中白色部分(以C/C++语言为例)。

首先是对象(Object)类，在我们所谓的对象中不仅包括C++语言中的类变量、C语言中的各种变量，还包括常量(ConstObject)、枚举子(Enumerator)、函数对象(FunctionObject)和模板函数(TemplateFunction)。

其次是域(Scope)类，纵观C和C++语言的语法，我们在程序模型中分成了以下八种不同的域，相同的甚至不同的域之间可以相互包含嵌套。

(1)文件域(FileScope)：一个独立的编译单元，包括了一系列的函数定义和变量定义；

(2)函数域(FunctionScope)：从函数的定义或声明的参数声明开始，到函数定义或声明结束为止；

(3)块域(BlockScope)：由一对花括号所包含的一段程序(块)，相当于是一个复合语句块；

(4) For域(ForScope)：从for语句的循环变量声明开始到for语句结束为止；

(5) 全局域(GlobalScope)：系统最外层的域，所有域都处于全局域中；

(6) 名空间域(NameSpaceScope)：对应于名字空间；

(7) 模板域(TemplateScope)：模板定义的范围内；

(8) 结构域(StructureScope)：类、联合、结构定义的范围内。

再次是型(Type)类，它包括：(1) 位域型(BitFieldType)、(2)数组型(ArrayType)、(3)基本型(BasicType)、(4)函数型(FunctionType)、(5)指针型(PointerType)、(6)引用型(ReferenceType)、(7)常量型(ConstType)、(8)用户自定义型(UserType)。

其中用户自定义型的类中还可继续划分为：①枚举型(EnumType)；②类型重定义型(TypedefType)；③结构型(StructureType)。结构型中还包括ClassType、StructType、UnionType和TemplateType。

最后是模板。C++程序由类和函数组成，模板也相应地分为类模板(ClassTemplate)和函数模板(FunctionTemplate)。

4 层次式程序表示模型

从结构示意图中，我们会发现这四者之间存在着内在的相互关联，反映了对象、域、型和模板的依赖关系。在构建程序模型时，结构图中所有的部分对应不同的类。ProgramData 类是所有元素的抽象基类；其次，Object、Scope、Type、Template 四个类直接继承 ProgramData 类；然后，结构图中的其余部分都作为一个类按照图中所示继承于不同的类；最后形成一个树状的类继承关系图。以下，分别对各个类的属性进行描述：

首先，Object 类中的属性 name 标识对象的名，type 用来标识对象的型，它指向某个 Type 类；scope_of_decl 表示定义或声明该对象的域，它指向某个 Scope 类。显然，名、型和域是对象的三个必不可少的重要属性。

其次，Scope 类的属性，除了域名之外，还有 upper_scope 标识本域的外层域。例如，对于文件域，其上层域是全局域，而全局域的上层显然为空。对于 Scope 类，还包括四个列表，分别是：scopes_list 保存本域中所有的直接下层域；types_list 保存本域中定义的所有的类型；template_list 保存了本域中定义的模板类；object_list 保存本域中定义的对象。本文提出的程序模型的层次性其实就在于域的层次性，即模型的不同层次对应于域的不同层次。

再次，Type 类的属性只有一个，即声明该型的域。scope_of_decl 指向 Scope 类。但这只是对于所有的 Type 类而言的，对于各个具体的 Type 类又分别有其不同的属性。例如对于数组型（ArrayType），它的属性还包括 element_type 和 array_size。因为，对于一个给定的数组，有了其元素的类型和元素的个数，该数组的类型也就唯一地确定了。

最后是模板，除了模板名以外，包括声明该模板的域（scope_of_decl）、模板的形式参数列表（parameters_list）和该模板所包括的模板域（template_scope）等等。

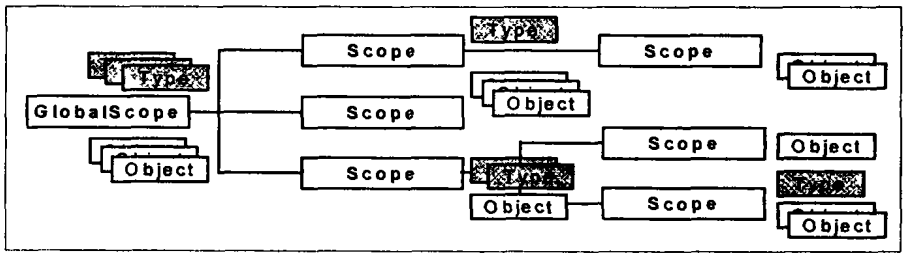

图2 层次式程序表示模型示例

综上所述，程序模型中的对象、域、型和模板是四个相对独立但又密切联系的部分。如图2所示，本文所提出的层次模型正是以不同层次的作用域为基础，以对象为内容，以型作为主线的一种递归嵌套的程序表示模型。

5 模型的实现及访问

直接从源程序构建该模型比较困难，因为该模型并非按照程序的正文顺序构筑。我们采用的方法是首先通过对程序进行词法分析和语法分析建立一棵基于源程序语句顺序的语法树[12]；然后访问语法树，按照程序的语法层次逐步构筑域的层次结构，并在不同域中检查和填写对象和类型信息以及对象和类型之间的依赖关系；并可把模型以文件的形式保存下来。同时，我们设定标准的访问接口来访问模型。

如图1所示，ProgramData 类是模型中所有类的基类。Object、Scope、Type、Template 四个类直接继承 ProgramData 类，其他的类又分别继承以上四个类。也就是说，模型中所有的类通过继承关系形成一个树型结构的模型，树根是 ProgramData 类，所有的类之间的联系使用指针来完成。由于在实际的模型没有构造之前，我们是不可能根据程序来判定指针指向的到底是什么类型的对象，因此统一使得指针的基本类型均为 ProgramData 类型，即任何一个模型中的指针指向的都是一个 ProgramData 类型的对象。至于在对不同程序建立的不同模型中存在的各种指针的基本类型究竟是什么，在模型的构建过程中可以通过动态定联机制来动态确定。

因此在访问该模型时将不考虑子类的具体情况，而在父类的抽象层次上按父类的接口进行程序的设计。即，在抽象层次上设计通用的访问程序，而在程序的运行中根据对象类型确定对模型进行访问的操作代码。根据模型中的抽象数据类型和继承层次结构，对于一个给定的 ProgramData 类型对象，使用虚函数的方法进行访问得到相关的信息。

6 程序模型的特点

程序模型侧重于反映程序中对象、域和型三者之间的依存关系。对于程序中出现的各式各样的对象来说，其作用域和数据类型无疑是最重要的两个方面。许多程序中出现的错误也正是由于在对数据的使用过程中没有考虑到域不同和类型不匹配引起的。虽然有的编译系统对对象使用中的类型不匹配可以进行有限的转换，有一定的容错功能，但是，这不是根本的解决之道。况且，有的时候这种转换也是很不安全的。本文所提出的程序表示模型是以作用域为基础，通过作用域把对象和类型紧密地联系起来。通过对本模型进行分析检测能够很方便地检测出以上所述的一系列问题。

程序模型能够全面详尽地展现程序中所有对象，并提供了标准的访问接口。由于模型中的类是相互继承的，因此，对于任何一个类的访问方法也是类似的。由于生成的程序模型是一棵节点树，我们可以通过对这棵树的访问来得到我们需要的信息。在程序模型的这棵树中每一个节点实际上都有一系列的其他的节点作为它的孩子。访问可以从根节点（GlobalScope）开始，进行深度遍历或者广度遍历直到获得相应的信息。我们还可以采用另一种方式。在模型的构建过程结束之后，使用列表将所有的相同类型的元素连接起来。例如：把模型中所有的 Enumerator 保存到 Enumerator_list 的列表中。在访问的时候可以按照不同的要求对不同的类的队列进行访问。这种访问方式避免了从根开始遍历，节约了访问程序的运行开销。这样，模型实际上提供了基于遍历的树型访问接口和

（下转第135页）

$r_t(x,y)$,$h_t(x)y$ 和 s_t 此处都是常数。给定边界条件 $t=0$,$\alpha=0$,求解该方程组,得 $t_t=t^*>0$和 $\alpha_t=\alpha^*$,使得要么 $\gamma^*\leqslant v$,要么 $t^*=s_t$。④修改每个例子的预测值为:$r_{t+1}(x,y)=r_t(x,y)+\alpha_t h_t(x)y$。⑤修改"剩余时间"$s_{t+1}=s_t-t_t$。

Until $s_{t+1}\leqslant 0$

输出:最终假设:

$$\begin{cases} \text{若 } p(x)\in[-1,+1],\text{则 } p(x)=erf\left(\dfrac{\sum_{t=1}^{N}\alpha_t h_t(x)}{\sqrt{c}}\right) \\ \text{若 } p(x)\in\{-1,+1\},\text{则 } p(x)=sign\left(\sum_{t=1}^{N}\alpha_t h_t(x)\right) \end{cases}$$

其中 erf 是"错误函数":$erf(a)=\frac{2}{\pi}\int_0^a e^{-x^2}dx$。

虽然 Boost-by-majority(简称 BBM)算法有优化特性,但它不是自适应的,因此很少得到实际应用。BrownBoost 算法是 BBM 算法的一个自适应扩展版本,对实际的学习问题是有用的。

AdaBoost 的成功是无可争辩的,但该算法很容易受到噪声的影响,因为它倾向于给噪声加较其它样本高的权重,导致后面的迭代步中产生的假设对噪声的过分匹配。AdaBoost 中,对一给定例子(x,y)如果大多数假设的预测是不正确的,那么将会有大负 margin 值,同时例子上的权重会无限制地快速增长。为避免这种情况,一些学者建议使用增长比 e^x 慢得多的 margin 的函数作为加权方案,如"Gentle-Boost"算法。然而,所有建议均无在 PAC 框架中定义的正规 Boosting 属性。上述与 AdaBoost 有关的实验性问题迫使我们重新审视 BBM。该算法赋给各例子的权重是 margin 的函数(BBM 产生其中所有假设都有相同权值的多数规则时,margin 是正确弱假设数目和迭代数目的线性组合)。然而,反映 magin 和权重关系的函数形式与其在 AdaBoost 中的形式有很大的差异。其原因是:BBM 是一个被优化的用于在预先设定的 Boosting 迭代步数内最小化训练错误的算法。该算法在接近预定的终点时,有大负 margin 值的例子将变得越来越不可能最终被正确标记。因此,放弃这些例子集中精力于具有小负 margin 值的例子上的 BrownBoost 算法效果应当更好。BBM 需要预先指定弱学习器上错误的上边界 $1/2-\gamma$ 和一"目标"错误 $\varepsilon>0$。BrownBoost 可以省去参数 γ。设置 ε 为0将使 BrownBoost 变为 AdaBoost。因此,可以说 AdaBoost 是 BrownBoost 的一种特殊情况。这就和 AdaBoost 在大噪声数据集性能很差这一事实在直观上相吻合。

参 考 文 献

1 Valiant L G. A theory of the learnable. Communications of the ACM, 1984, 27(11): 1134～1142
2 Kearns M J, Vazirani L G. Learning Boolean formulae or finite automata is as hard as factoring: [Technical Report TR-14-88]. Harvard University Aiken Computation Laboratory, Aug. 1988
3 Kearns M J, Vazirani L G. Cryptographic limitations on learning Boolean formulae and finite automata. Journal of the Association for Computing Machinery, 1994, 41(1): 67～95
4 Schapire R E. The strength of weak learnability. Machine Learning, 1990, 5(2): 197～227
5 Freund Y. Boosting a weak learning algorithm by majority. Information and computation, 1995, 121(2): 256～285
6 Freund Y, Schapire R E. A decision-theoretic generalization of online learning and an application to boosting. Journal of Computer and System Science, 1997, 55(1): 119～139
7 Dietterich T G, Bakiri G. Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research, 1995, 2: 263～286
8 Schapire R E, Singer Y. Using output codes to boost multiclass learning problems. In: Machine Learning: Proc. of the Fourteenth Intl. Conf. 1997. 313～321
9 Schapire R E, Singer Y. Improved boosting algorithms using confidence-related predictions. In: Proc. of the eleventh Annual Conf. on Computational Learning Theory, 1998. 80～91
10 Friedman J, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting: [Technical Report]. 1998
11 Freund Y. An adaptive version of the boost by majority algorithm. In: Proc. of the Twelfth Annual Conf. on Computational Learning Theory, 1999
12 刁力力,等. 数据挖掘与组合学习. 计算机科学[J], 2001, 28(7)
13 涂承胜,刁力力,陆玉昌. BoostingAdaBoost 系列代表算法. 计算机科学, 2003, 30(3)

(上接第120页)

基于列表的线性访问接口;通过标准的访问接口,任何符合该接口的高层应用模块可从中检索需要的信息。

程序模型的内聚度高,可扩充性强。程序模型是以程序中的对象为基本的单元,以域的层次来划分对象,以对象的型为主要的组成部分来构建,它还特别包含了模板的信息。模型中使用指针相互关联,使得各个部分之间既相对独立又相互依赖,形成一个紧密的整体。同时,指针的使用可以使得增加或减少单元的操作非常方便,对整个模型的影响比较小。当针对不同的语言构建模型时,可以根据不同语言的语法特点相应修改模型中的成员。

结论 基于对象、域和型的层次式程序表示模型是一种能够详尽描述程序结构的一种中间表示形式,它把程序中所有对象的作用域、类型等信息紧密地关联起来。该模型采用了层次式的结构,有利于通过计算机自动地分析和理解程序的结构;并且它提供了标准的访问接口,任何符合该接口的应用模块均可从中抽取所需要的相关信息。这样,在该模型的基础上,我们可以构筑很多有意义的应用。

参 考 文 献

1 Eckel B. C++编程思想. 北京:机械工业出版社, 2000
2 Pressman R S. 软件工程--实践者的研究方法(第四版). 北京:机械工业出版社, 1999
3 刘超,等. 可视化面向对象建模技术. 北京:北京航空航天大学出版社, 1999
4 张幸儿. 计算机编译原理. 北京:科学出版社, 1999
5 Binder R V. 面向对象系统的测试. 北京:人民邮电出版社, 2001
6 Hughes C. 掌握标准C++类. 北京:人民邮电出版社, 2000
7 Pratt T W. Programming Languages: Design and Inplementation Prentice-Hall International, Inc 1996
8 Norman R J. Object-Oriented Systems Analysis and Design Prentice-Hall International, Inc 1996
9 Martin R C. Pattern Languages of Program Design Addison Wesley Longman, Inc. 1998
10 Peter B J. Introduction to Compiling Techniques A First Course using ANSI C, LEX and YACC. McGRAW—HILL Book Company, 1990
11 Aho A V, Sethi R, Ullman J D. Compilers: Priciples, Techniques, and Tools. Addison-Wesley, 1986
12 赵洋,蔡志旻,潘金贵. 基于EPOM的程序可视化表示系统的设计与实现. 上海:计算机工程(已录用于2002年第7期)