程序验证方法 逻辑模型 数理逻辑 14

计算机科学 1995 Vol. 22 №. 3

58-60

关于程序验证方法的讨论

周 青 TP31

（中山大学计算机软件研究所 广州 510275）

摘 要 The researching in the program correctness verification has been taken decades, but so far as we know no substantial progress has been made. In this paper, we attempt to discuss the problem, including a discussion on the weakness in traditional methods and existance of a new method.

关键词 Syntactical function, Semantical function, Program verification.

一、引 言

自从 1967 年 Floyd 发表其论文“给程序赋予意义”[4]以来，程序自动验证工具的研究已持续了数十年。Floyd 在文[4]中提出了用“断言式方法”证明程序的正确性。其具体作法是：在程序流程图的每一条边上附上一个谓词公式（称为断言）。对于循环，即流程图中的回路，他定义了一个切点，并在切点上设置一个断言。然后定义：一个程序是正确的，如果：(1)对于程序流程图中的每一条边，当程序运行到该条边时，其所附的断言为真；(2)对于循环，若回路开始执行时其切点上的断言为真，则当循环回复到该切点时其断言仍应该为真，即所谓的“循环不变式”。

1969 年，Hoare 发表了“计算机程序设计的公理基础”一文，建立了所谓的“Hoare”逻辑。这是一种含有程序公理和推理规则的形式逻辑推理系统。由于这种逻辑系统可在计算机上实现，人们便用它来证明程序的部分正确性。这种程序正确性验证方法称为“公理式方法”。

尽管在程序正确性验证方法的研究中，这两种方法在当今世界上占统治地位，但是，就笔者所知，到目前为止还没有取得令人满意的进展，其主要原因在于传统的方法有难以克服的困难。对于公理式方法，由于有意义的公理系统通常包括不止一条公理，根据数理逻辑的理论研究，我们知道不存在一个能行的机械算法可以使计算机能从公理系统中选择合适的公理来证明所需的结论，因此，这种方法的计算复杂度是指数形式的。更为重要的是，当程序不正确时，计算机会一直工作下去而不停止。（根据数理逻辑的研究，一个一阶谓词逻辑系统的定理集合是递归可枚举集，即半可计算集。）对于断言式方法，主要困难就是如何选择正确的断言，我们无法设计一个通用的算法由计算机来完全承担。这样，在验证的过程中，人的提示将无法避免，程序的正确性验证工作就会大大地依赖于操作人员的实际工作能力，而且也会极大地增加验证工作所花费的时间。同时，在断言的证明过程中还会遇到类似于公理式方法中所遇到的困难。

本文作者认为：程序正确性验证仅利用语法方面的信息是无法完成的，必须同时考虑程序的语义。人们总是为着解决某种问题而编制程序的，因而任何程序中都带有相当多的语义方面的信息，只有充分利用这些语义方面的信息才有可能达到程序正确性验证的目的。

二、程序正确性：语法函数和语义函数

为了便于讨论，首先简要地介绍一下关于模型的概念。

设 T 是一个以 L 为其形式语言的一阶逻辑系统。如果 M 满足如下条件则称 M 是 T 的一个模型：

①存在一个非空集合 |M|，（称为 M 的全域）|M| 中的元素称为 M 的个体；

②存在一个由 |M| 到 |M| 的函数集合；

③存在一个 |M| 上的谓词集合；

④存在一个映射 φ：L→M 使得；

1. 对于 L 中的每一个常元符号 a，φ(a) 是 M 中的个体；

2. 对于 L 中的每一函数符号 f，φ(f) 是 M 上的函数；

3. 对于 L 中的每一个谓词符号 p，φ(p) 是 |M|

上的谓词；

4. 对于 T 中的每一条公理 A，φ(A)在 M 中为真，这里，φ(A)是根据以上的初始定义并按通常的归纳定义方式定义的。

由于 M 使得 T 中的每一条公理为真，可以立即得到：

1. 如果 $\vdash_\tau a \neq b$，则 $\varphi(a) \neq \varphi(b)$；

2. 如果 $\vdash_\tau f \neq g$，则 $\varphi(f) \neq \varphi(g)$；

3. 如果 $\vdash_\tau p \neq q$，则 $\varphi(p) \neq \varphi(q)$。

所以，φ 是一对一的。

为了今后叙述方便，假定每一个 M 中的个体在 L 中都有一个名字。这样，φ 就是一个从 L 到 M 的一对一的满射。

现在，我们回到程序正确性验证。当我们编制程序时，希望在输入数据并使程序在计算机上运行之后它将输出其运行所得到的结果。于是，每一个程序都可以看作是一个从输入符号集合到输出符号集合的函数。由于计算机只能识别符号串而不能理解符号串所表达的实际含义。所以程序本身只能是语法意义上的函数。另一方面，人们总是为某种目的给计算机输入具有实际含义的数据，并希望在程序运行之后会给出所期望的具有实际含义的结果。因此，在人们的头脑中，程序是一个具有实际涵义的函数，即是一个从符号的实际含义集合到符号实际含义集合的语义意义上的函数。这样，在程序和人们头脑之间有图 1 所示的映射 φ：

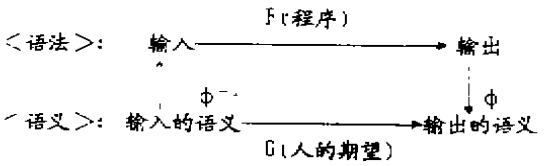

图 1

根据我们对模型的描述，很明显，人们头脑中对程序和数据的语义解释是程序的一个模型，即语义模型。这样，我们可以立即得到：如果程序的语义解释和其程序设计人员所期待的相一致，则该程序就是正确的，反之亦然，即一个程序是正确的，当且仅当它的语法函数和语义函数是一致的。于是，我们可以把程序正确性验证用图 1 中的符号来表述为：所谓程序的正确性验证就是要证明 $\varphi(F)=G$。

以下将要讨论以这种思想为基础而产生的程序正确性方法的具体作法。

三、语义函数的生成

如上所述，本文作者把程序看作是语法函数，而把程序的语法函数和语义函数的一致性作为程序正确性验证的标准。给定一个已经编制好了的程序 F，我们希望证明它符合其设计目的。

为了让计算机对程序的语法函数和语义函数的一致性作出判断，我们首先需要对程序的语义作出一定的描述，即必须先把程序的语义含义输入到计算机中，然后计算机才能进行判断。由于我们在编制程序之前已对程序的目的和算法有了构想，这并不是一个很过分的要求。

同时，因为我们希望由计算机来进行程序正确性验证，计算机应能根据程序而生成它的语义形式，并可以把它和所输入的程序语义含义进行比较，从而判断它们是否一致。本文将主要讨论计算机如何根据程序来生成其语义函数，至于判断它与所输入的程序语义含义是否一致的问题将留待以后讨论。

在程序的编制过程中，我们可以使用：Ⅰ）由程序设计语言提供的各种指令；Ⅱ）循环语句；Ⅲ）调用子程序。以下分别说明计算机如何生成这些技术的语义函数。

显然，如果程序 F 只包含一条指令，则该指令的说明就是它的语义含义，因而它的语义形式是明显的。

如果程序 F 是由一串按顺序执行的指令组成，其语义形式亦不难由计算机生成，因为计算机仅需把这些指令的语义按程序的顺序分别列举出来即可。

生成循环语句的语义函数相对来说比较困难一些。在各种程序设计语言中，循环语句有许多不同的表达方式，为了叙述的方便，本文将不对这些不同的表达方式进行分别的讨论，而把它们都看作是同一类运算，并以 for…to…do…为例来进行讨论。但是，这里所述的方法只需作一些简单的修改也能适用于其它形式的循环语句。

对于程序正确性验证而言，主要有三种不同的循环语句需要进行分别考虑。

①假定程序 F 的语义函数已经确定，而我们所讨论的循环语句具有如下形式时：

for k=0 to n do

$F(x_1,\cdots,x_m)$

next k

由于循环变元 k 不在程序 F 中作为变元出现，这类循环语句所做的只是重复执行程序 $F(x_1,\cdots,x_m)$ 而已（比如求 2"）。显然，这种循环语句的语义函数是：

"重复 n+1 次执行 $F(x_1,\cdots,x_m)$"。

②如果循环变元 k 是程序 F 的一个变元，即考虑如下形式的循环语句：

for k=0 to n do

$F(k,x_1,\cdots,x_m)$

next k

则它的语义函数是：

"从 k=0 开始执行 $F(k,x_1,\cdots,x_m)$ 直到 k=n。"

③最后，当我们遇到具有形如：

x=a

for k=0 to n do

$F(k,x,x_1,\cdots,x_m)$

$x=F(k,x,x_1,\cdots,x_m)$

next k

的循环语句时（例如求 n!），它是一个用递归方式定义的函数，其语义函数为：

"求 $f(n,x_1,\cdots,x_m)$，其中，$f(0,x_1,\cdots,x_m)=a$；$f(k,x_1,\cdots,x_m)=F(k,f(k,x_1,\cdots,x_m),x_1,\cdots,x_m)$。"

由于以上各种不同的循环语句形式中，仅从语法形式上就能判断它们的语义函数的类型，所以生成它们的语义函数是可以由计算机来完成的。

对于调用子程序，只需将子程序看作是一个单个的宏指令即可容易地知道它的语义函数是可以由计算机来完成的

综上，生成程序的语义函数是可以由计算机来完成的。由于我们的讨论显然不受具体的程序设计语言的限制，所以可以得出如下结论：对任何程序设计语言 L 存在一个程序，它可以生成用 L 语言编制的任何程序的语义函数。

以上所讨论的是如何生成程序的语义函数的方法。从本文的讨论可以看出本文所提出的程序正确性验证方法和传统的方法比较起来有如下主要优点：

1. 不会产生计算机永不停机的情况。如上所述，一阶逻辑系统的定理集是递归可枚举集，因而对于不正确的程序，传统的验证方法会由于试图证明它是正确的而一直工作下去。因为即使不正确的程序也有其语义含义，故本文所述的方法不会产生此类问题。

2. 可以充分利用程序所带的语义方面的信息。由于传统的程序正确性验证方法试图利用公理系统来证明程序或断言，一旦把程序输入，验证程序将调出公理和推理规则以求证明该程序的正确性，于是，该程序就和它的语义含义脱离了关系。本文所述的方法却非如此，我们紧紧抓住程序的语义含义并与人们对程序所期望的结果相比较，从而充分利用了程序的语义信息。

3. 可以简化程序的验证过程。在程序中通常包括一些仅为编程需要的语句，它们没有特别的涵义，传统的程序正确性验证方法由于要进行形式证明而必须处理它们。因为程序的语义函数不必考虑这类语句，因而本文所述的方法也就不存在这样一类问题了。

参考文献

[1] Apt, K. R., Ten Years of Hoare's Logic, a Survey-Part Ⅰ, ACM Trans. on Programming Languages and Systems 3, 1981

[2] Apt, K. R., Ten Years of Hoare's Logic, a Survey-Part Ⅱ, Theoretical Computer Science 28, 1984

[3] Davis, M., Computability, 1974, A Lecture in New York University

[4] Floyd, R. W., Assigning Meanings to Programs, Proc. Symposium on Applied Mathematics, American Mathematical Society, Vol. 19, 1967

[5] Hoare, C. A. R., An Axiomatic Approach to Computer Programming, CACM, Vol. 12, No. 10, 1969

[6] Kleene, S. C., Introduction to Metamathematics, North Holland, Amsterdam, 1952