Research on
Program Correctness

程序正确性证明方法

武 斌 著

上海财经大学出版社

Research on

Program Correctness

程序正确性证明方法

武 斌 著

上海财经大学出版社

本书由上海财经大学浙江学院发展基金资助出版

图书在版编目(CIP)数据

程序正确性证明方法/武斌著.—上海:上海财经大学出版社,2018.12
ISBN 978-7-5642-3169-9/F·3169

Ⅰ.①程… Ⅱ.①武… Ⅲ.①程序设计 Ⅳ.①TP311.1

中国版本图书馆 CIP 数据核字(2018)第 271163 号

□ 责任编辑 柳萍萍
□ 封面设计 张克瑶

程序正确性证明方法

武 斌 著

上海财经大学出版社出版发行
(上海市中山北一路 369 号 邮编 200083)
网 址:http://www.sufep.com
电子邮箱:webmaster @ sufep.com
全国新华书店经销
江苏凤凰数码印务有限公司印刷装订
2018 年 12 月第 1 版 2018 年 12 月第 1 次印刷

710mm×1000mm 1/16 8.5 印张(插页:2) 124 千字
定价:39.00 元

前　言

程序验证是计算机程序设计领域的前沿研究课题，如何保证程序正确性是计算机科学的一个重大挑战. 近年来，随着符号计算理论的不断完善和程序验证中使用精确无误差的数学方法的要求，使用符号计算理论来解决程序验证中的相关问题被认为是一种有效的途径. 本书在前人研究的基础上，利用符号计算的思想和方法研究了程序验证领域的三个基本问题：循环不变式生成、程序终止性分析以及前置条件生成.

循环不变式在程序的部分正确性验证中起着非常重要的作用，如何生成循环不变式也是程序验证领域的挑战之一. 本书主要研究了多项式循环程序的不变式生成问题. 首次将有限点集消去理想的思想和方法应用于多项式循环程序的不变式生成，设计了一个多项式时间复杂度的循环不变式自动生成算法，可生成多项式等式型循环不变式.

程序的终止性分析问题也是长期以来为众多计算机科学家所关注的问题之一. 本书主要研究了一类带有非线性循环条件和线性赋值的循环程序的终止性分析问题. 通过计算线性赋值矩阵的约当标准型，确定循环条件在循环次数充分大时的符号，将这类循环程序的终止性分析问题转化为判断参系数半代数系统有无实解的问题. 如果参系数半代数系统中的左端函数个数有限或者左端函数都具有整数周期，则这类非线性循环程序的终止性问题是可判定的.

另外一个值得研究的问题是如何计算合理的前置条件，使得循环程序在满足该条件的前提下是终止的. 本书基于一阶常系数差分方程组的求解技术，设计了一个高效、实用的前置条件生成算法. 将程序的循环赋值语句转化为程序变量关于循环次数的差分方程组，计算差分方程组的闭形式解. 然后将闭形式解代入循

环条件，在循环次数充分大时，判断循环条件的符号，进而生成循环程序合理的前置条件. 针对线性赋值程序给出了一个高效、实用的前置条件自动生成算法. 进而，对于可求出闭形式解的非线性赋值循环程序以及运算可交换的多分支循环程序，也做了相应研究.

研究结果表明，符号计算是验证程序正确性的一种行之有效的方法. 我们期待将符号计算中的一些经典算法更深入、广泛地应用到程序验证，并集成、研发新的有前途的验证工具.

本书的编写得到上海财经大学浙江学院发展基金、浙江省高等学校中青年学科带头人培养对象培养资助经费和金华市 321 人才工程培养经费的资助.

由于作者水平和能力有限，书中难免出现不当之处甚至错误，恳请读者批评、指正.

武 斌

2018 年 5 月

目　录

第一章

绪　论

程序验证(program verification)是研究程序正确性的理论，是计算机程序设计领域的传统研究课题，但迄今为止，对常见的绝大多数程序而言，实现完全的正确性验证还是很困难的[45]. 然而，随着计算机应用范围的日益扩大和计算机程序设计语言的迅速发展，如何保证程序的正确性、提高软件的可信度一直是计算机科学界高度关注的一个重要问题，也是推动计算机科学发展的主要动力之一.

一段程序是正确的，是指这段程序能正确无误地完成程序设计时所期待的功能. 目前，为了检测一个程序是否正确地实现了预定的目标，通常使用两种方法：程序测试(program testing)和程序验证. 程序测试通常是规定一些初始数据，试验性地执行这个程序，测试其是否能产生所要的答案. 如果发现有误，就检查和修改所编写的程序，直至对所有规定的初始数据都能产生预期的结果. 但是，程序对不同的初始数据的加工过程是不同的，而初始数据的取值范围往往又十分广泛，因此，使用测试方法穷尽程序的各种可能加工过程以确保程序的正确性，几乎是不可能实现的. “测试方法只能发现程序的错误，而不能确保程序无误。”[75]

而程序验证是研究程序正确性的理论，是研究如何使用数学方法严格证明程序所有可能的初始数据都是符合其预定目标的，因而是正确无误的.

虽然程序测试不能保证待测程序没有错误，但这并没有阻碍它在实际中的应用. 程序测试是面向目标代码(object code)的，因此程序员可以利用测试方法很快找到错误并加以改正. 面对检测需求不明确的软件，程序测试也体现了其速度上的优势. 相对测试而言，通常程序验证的工作量更大，但是验证技术也具有更多的优点:(1)应用程序验证技术，我们可以在更深的层次洞察程序，并且能够帮助我们真正理解程序的思想以及它的局限性[46]；(2)程序验证技术可以更好地支持面向目标的程序开发(goal-oriented program development)[84]。由此可见，程序验证是保证程序正确性的一个基本而重要的手段，也是计算机软件工程方法和技术中不可或缺的重要组成部分.

由于 E. Dijkstra, C. A. R. Hoare, R.Milner 以及 A. Pnueli 等一大批计算机科学家关于程序验证理论的出色工作，使得程序验证这一研究方向取得了广博的发展，也产生了大量具有实用价值的理论和技术. 到 20 世纪 80 年代，由于程序验证中数学证明的复杂性，研究热度有所下降，甚至许多悲观的研究者认为程序验证的理论和方法没有应用前途.

符号计算作为一种精确、无误差的计算方法，正好符合程序验证使用正确无误的数学方法的要求. 目前，随着符号计算理论与方法的迅猛发展，诸如量词消去[56, 57, 150]、多项式代数[120, 121, 123, 124, 146]、Gröbner 基[23, 31, 42, 43, 44, 71, 72, 81]、参系数半代数系统的实解分类[13, 20, 51, 157, 158, 163, 164]等，为程序验证提供了强有力的理论基础，验证技术及其应用的研究也随之活跃起来.

众所周知，我们正处在信息时代，随着计算机应用范围的日益扩大，软件系统不仅仅出现在航空、医药、军事等安全攸关领域，我们的日常生活也与软件产品息息相关. 但是，软件产品中的“bug”层出不穷，有些还导致了严重后果. 如何保证程序的正确性、提高软件的可信度就显得尤为重要. 甚至，国际著名的计算机科学家、图灵奖得主 C. A. R. Hoare 提议将程序验证作为计算机科学中的一个重大

挑战性问题，希望能像人类基因组计划那样，通过国际合作以求在该方向取得重大进展[91]. 在如此严峻的形势下，将符号计算作为一种新的思想和方法应用于程序验证研究，期待验证技术能有所创新. 正是基于上述考虑，本书将结合符号计算的思想和方法来展开对程序验证理论和技术的研究.

本章将从程序正确性研究方法、程序正确性研究现状、符号计算及其应用等方面进行简要综述.

1.1 程序正确性研究方法概述

本节将介绍几种目前较为常用的程序正确性检测技术，并且较详细地叙述在后续几章中涉及的重要概念和定义. 特别地，重点介绍目前最流行的程序正确性证明和验证的方法之一——公理化方法.

1.1.1 程序测试

测试是目前工业界普遍采用的提高软件质量的重要手段，也是软件工程学术界研究的主要内容之一. 虽然目前有不少工具能帮助用户统计测试过程中的各种信息，但是测试用例的设计(或者说测试数据的生成)是难题之一. 如何在给出比较少的测试用例的条件下达到比较高的覆盖度，将是一个长期研究的课题.

1.1.2 模型检验(model checking)

这类方法主要针对可归结为有限状态的程序. 这里所谓的“状态”是指程序变量的一种取值. 例如，假设程序中共有两个变量：布尔变量 b 以及取值范围为[1…10]的整型变量 n。那么该程序可以有 20 个状态：$\langle b = \text{FALSE}, n = 1\rangle$，$\langle b=\text{TRUE}, n=1\rangle, \cdots, \langle b=\text{FALSE}, n=10\rangle, \langle b=\text{TRUE}, n=10\rangle$. 通过枚举所有的状态，可以检查程序是否具有某些性质. 一般来说，程序可表示为状态机(赋值语句对应于状态迁移)，其性质可用时序逻辑公式来表示. 用模型检验方法可以自

动地验证很多通信协议和并发控制算法的正确性.

1.1.3 静态分析(static analysis)

静态分析是指不执行程序,通过类型推导(type inference)、抽象解释(abstract interpretation)等手段检查程序中是否具有某种错误(比如内存泄漏). 有些工具也使用特定的规则对程序进行检查. 比如在多线程程序中要求在使用共享变量时遵守"使用前先加锁,使用后解锁"的规则.

1.1.4 公理化方法

上面介绍的三种程序正确性检测方法都是以发现错误为主要目标的,接下来重点介绍一种以证明程序正确性为目标的验证技术——公理化方法.

公理化方法是目前最流行的程序正确性证明方法之一. 这种方法最早是在1967年由R. W. Floyd基于流程图的验证而提出的[80]. 随后,C. A. R. Hoare经过不断发展和完善,于1969年提出程序验证的公理系统(当时主要是针对WHILE型程序)[90]. 这个系统为使用严格的数理逻辑推理计算机程序的正确性提供了一组逻辑规则. 这种方法依赖于对程序的每个成分定义断言(assertion),而我们可以应用这些断言,建立程序与其语义(semantic)之间的联系,换言之,可以建立程序与程序规约(program specification)所表达的需求行为之间的联系.

程序(program)描述了一个确定的输入状态如何转化为一个所需求的输出状态,而程序规约准确地描述了程序所应实现的具体功能. 为了更好地利用规约来说明程序,使用Hoare三元组(Hoare triple)来表示程序的执行. Hoare三元组具有如下形式:

$$\{P\}S\{Q\} \tag{1.1}$$

其中,S 是一段程序;P 是一个逻辑公式,被称为前置条件(precondition)或 S 的输入断言(input assertion);Q 是一个逻辑公式,被称为后置条件(postcondition)或 S 的输出断言(output assertion).

直观上，一段程序是正确的，是指它能满足给定的输入、输出关系，即满足程序规约. 这里，给出程序正确性的定义：

定义 1.1.1([33]) Hoare 三元组$\{P\}S\{Q\}$是部分正确的(partial correctness)，是指如果程序S执行前满足断言P，则当程序终止后，它一定蕴含断言Q.

定义 1.1.2([33]) Hoare 三元组$\{P\}S\{Q\}$是完全正确的(total correctness)，是指如果程序S执行前满足断言P，并且程序S终止，则它的最终状态一定蕴含断言Q.

由于一段程序S并不一定总是终止的，从上述定义可知，程序的部分正确性是在假设程序终止的前提下证明的，而

程序的完全正确性＝程序的部分正确性＋程序终止性.

【例 1.1.3】 考虑 Hoare 三元组$\{x=5\}\ x:=2x\ \{x>0\}$. 在本例中，

- $x=5$是前置条件；
- $x:=2x$是一段程序；
- $x>0$是后置条件.

显然这个三元组是部分正确的，也是完全正确的. 因为如果$x=5$，并且执行x乘以 2，我们得到$x=10$，这是肯定蕴含$x>0$的.

【例 1.1.4】 给定两个自然数x和$y(>0)$，计算x除以y的商 quo 和余数 rem.

计算 quo 和 rem 的程序的前置条件可以理解为“给定两个自然数x和$y(>0)$”. 如果规定取值范围是自然数$\mathbb{N}$，则前置条件可以用如下逻辑公式表示：

$$(x\geqslant 0)\wedge(y>0).$$

后置条件可以理解为“x除以y的商 quo 和余数 rem”，根据商和余数的性质，后置条件可以用如下的逻辑公式表示：

$$(quo\cdot y+rem=x)\wedge(0\leqslant rem)\wedge(rem<y).$$

计算x除以y的商 quo 和余数 rem 的程序代码如下：

$quo:=0;\ rem:=x$

while $y\leqslant rem$ **do**

$rem:=rem-y;$

$quo:=quo+1.$

end while

因此，这段程序的 Hoare 三元组表示如下：

$quo:=0;\ rem:=x$

while $y\leqslant rem$ **do**

$\{(x\geqslant 0)\wedge(y>0)\}$ $rem:=rem-y;$ $\{(quo\cdot y+rem=x)\wedge(0\leqslant rem)\wedge(rem<y)\}.$

$quo:=quo+1.$

end while

1. 循环程序(loop program)部分正确性验证

众所周知，Hoare 公理化方法最具挑战性的部分是验证循环程序的正确性，我们要建立断言来描述循环的具体执行，将建立的断言称之为循环不变式(loop invariant)，即在循环体每次执行前后都为“真”的断言.

循环不变式在程序的部分正确性验证中起着非常重要的作用，它可以提醒程序员在修改程序时避免违反程序正确性的约定. 反之，如果缺少表达形式简洁、明确的循环不变式，程序员在修改程序时很容易产生错误. 因此，如何自动生成循环不变式是程序验证领域的挑战之一.

下面，将用 Hoare 三元组的形式来定义循环程序

While b **do** S

的循环不变式：

定义 1.1.5([103]) 断言 I 是 Hoare 三元组$\{P\}$ **While** b **do** S $\{Q\}$的循环不变式，是指 I 满足下列条件：

(1)$P\Rightarrow I$：循环不变式满足前置条件；

(2)$\{I\wedge b\}\ S\ \{I\}$：循环不变式满足循环的每次执行；

(3)$I \wedge \neg\, b \Rightarrow Q$:循环不变式和循环退出条件蕴含后置条件.

【例 1.1.6】 例 1.1.4 中的 Hoare 三元组具有如下形式的循环不变式:

$$(quo \cdot y + rem = x) \wedge (0 \leqslant rem) \wedge (0 < y) \wedge (x \geqslant 0).$$

通常,一个循环程序有很多循环不变式,所有这些循环不变式都能帮助我们证明程序的需求性质.

2. 循环程序完全正确性验证

为了验证程序的完全正确性,必须在证明部分正确性的基础上进一步证明每段程序的终止性,尤其是含有迭代的程序,比如循环程序. 目前,证明程序正确性最常用的方法之一就是计算程序的秩函数(ranking function).

定义 1.1.7([101]) 函数 r 是 Hoare 三元组 $\{P\}$ **While** b **do** S $\{Q\}$ 的秩函数,是指 r 满足下列条件:

(1) r 是定义在每次循环执行后的程序变量上的整数值函数;

(2) $I \wedge b \Rightarrow r > 0$:能够进入循环(循环条件 b 为真)并且满足循环不变式,则 $r > 0$;

(3) $\{I \wedge b \wedge r = R\} S \{r < R\}$ 每次循环执行后,函数 r 的值总是递减的(R 是常数).

由上述定义,可以看出,只要秩函数存在,循环程序的循环次数总是有限的,因此是终止的.

【例 1.1.8】 对于例 1.1.4 中的 Hoare 三元组,rem 可以作为其秩函数.

将整数值函数作为秩函数,有时并不能很好地证明程序的终止性,已经有很多更加巧妙的方法被应用于终止性的证明[34, 40, 41, 63, 64, 113, 141]. 此外,对于函数调用和递归程序,这类程序的终止性往往被单独讨论[93].

3. 最弱前置条件和谓词转换

上述已经介绍了两种基于 Hoare 逻辑的程序正确性验证方法. 下面,将介绍程序验证中的另外一种重要方法,即 E. W. Dijkstra 于 1976 年提出的最弱前置条件策略(weakest precondition)[76]. 这种方法与 Hoare 公理化系统既有所联系,又

有所区别.

定义 1.1.9 给定逻辑公式 P 和 R，如果 $R \Rightarrow P$，则称 P 比 R 是更弱的(weaker)逻辑公式.

从定义可以看到，如果逻辑公式 P 相对于 R 是更弱的，则 P 比 R 包含了更多的可能状态.

当验证带有后置条件或用户需求明确的程序时，计算基于 Hoare 逻辑的最弱前置条件是非常有用的. 给定一个只有后置条件而没有前置条件的 Hoare 三元组：

$$\{?\}S\{Q\}.$$

通常，存在很多的前置条件 P 保证上述 Hoare 三元组是完全正确的. 但是，只存在唯一的前置条件 P 满足下列性质：P 包含了程序 S 终止的所有可能初始状态，并且程序执行结束后的状态蕴含后置条件 Q. 这样的前置条件 P 叫作最弱前置条件.

定义 1.1.10 给定程序 S 和其后置条件 Q. 称断言 P 为程序 S 和后置条件 Q 的最弱前置条件，是指对于保证 Hoare 三元组 $\{R\}\ S\ \{Q\}$ 完全正确的所有断言 R，都有

$$R \Rightarrow P.$$

而且 Hoare 三元组 $\{P\}\ S\ \{Q\}$ 也是完全正确的.

将程序 S 和后置条件 Q 的最弱前置条件记作为 $wp(S,Q)$.

换言之，$wp(S,Q)$ 是一个满足下列性质的谓词：$wp(S,Q)$ 描述了保证程序 S 终止的所有初始状态，并且程序执行结束后蕴含状态 Q. 因此，可以记作：$\{wp(S,Q)\}\ S\ \{Q\}$.

【例 1.1.11】 给定程序 S：$y:=x*x$ 和后置条件 Q：$y \geqslant 4$. 考虑计算关于程序 S 和后置条件 Q 的前置条件 P.

前置条件 $x \geqslant 2$ 能够保证程序 S 执行后蕴含状态 Q. 更强的前置条件可以是 $x \geqslant 3$、$x=3$ 等. 但是，它的最弱前置条件是：

$$wp(S,Q)\equiv(x\leqslant -2)\vee(x\geqslant 2).$$

应用最弱前置条件的方法来验证 Hoare 三元组$\{P\}\ S\ \{Q\}$，首先，要计算出关于程序 S 和后置条件 Q 的最弱前置条件

$$wp(S,Q),$$

接下来，要证明

$$P\Rightarrow wp(S,Q).$$

不幸的是，如果程序 S 包含循环程序，其最弱前置条件可能不具有明确的表达形式. 甚至，即使有明确的表达形式，最弱前置条件的计算也是非常困难的. 因此，如何有效地计算最弱前置条件也是程序验证领域的挑战之一[82].

可以将 wp 看作一个函数，一个谓词(后置条件) 经过 wp 作用，得到另外一个谓词(最弱前置条件). 因此，wp 又被称为谓词转换(predicate transformation). 有关谓词转换更详细的讨论，请参见文献[76].

1.2 程序正确性研究概况

根据上节的介绍可以看出，循环不变式、程序终止性以及最弱前置条件在验证程序正确性时起着非常重要的作用. 基于此，本书也将着重讨论上述三个问题. 本节将分别综述程序正确性理论和方法在上述三个方面的研究概况.

1.2.1 循环不变式研究概况

早在 20 世纪 70 年代中后期，循环不变式的自动生成技术就已经为众多科学家深入研究. S. M. German 和 B. Wegbreit[83, 148, 149] 以及 S. Katz 和 Z. Manna[99] 提出了利用求解递推方程(recurrence equation)的方法来生成循环程序的不变式. 虽然这种方法可以生成功能强大的不变式，但是由于求解递推方程的复杂性，此方法很难推广到非平凡的循环程序.

此后，M. Karr 和 P. Cousot 等人将抽象解释的技术应用于线性等式型和线

性不等式型不变式的生成[68, 70, 98]. 随后，M. Karr 的工作为 M. Müller-Olm 和 H. Seidl等人所推广[122, 124]，针对线性赋值循环程序，利用线性代数的相关技巧，生成了给定次数的多项式型不变式. A. Tiwari 等人还结合自顶向下(top-down approach) 和自底向上(bottom-up approach)技术自动生成循环不变式[144]. P. Cousot小组基于抽象解释技术开发了静态程序分析工具 ASTRÉE[4, 74]，该工具既能进行程序终止性分析，又能进行与不变量生成相关的程序验证. 并且，ASTRÉE 已经被广泛应用到工业软件的验证中，例如空客 380 系列飞行控制程序等.

然而，应用抽象解释技术的最大弊端是为了保证计算过程的终止而引入了 Widening 算子，由于比较粗糙的抽象而导致产生平凡的不变式. 此外，T. Bultan 等人结合 Presburger 算法生成程序不变式[47]. T. A. Henzinger 等人还将抽象解释技术应用于混成系统(hybrid system)不变式的生成[32, 88].

近期，基于约束求解的循环不变式自动生成技术(constraint-based invariant generation)成为不变式生成的主要方法之一. 应用这种方法，需要预先假定一个未知系数的不变式(多项式等式或多项式不等式)模板(template). 然后，根据不变式的定义，生成关于该模板系数的约束. 最后，通过求解约束，得到预先给定模板形式的不变式. 这种方法的优点在于，没有引入 Widening 算子，因而不会仅仅只生成平凡的不变式. 而且这种方法在理论上能够生成不高于模板次数的所有循环不变式(等式型和不等式型). 但是，在实际操作中，该方法也存在很大的弊端，主要原因是求解约束问题的复杂性，尤其是多项式约束求解问题. 目前，大都采用 Gröbner 基算法和量词消去技术求解约束问题，例如，S. Sankaranarayanan等人通过 Gröbner 基算法求解约束[133]，D. Kapur 利用量词消去方法求解约束[96]. S. Sankaranarayanan等人还将基于约束求解的技术应用于 Petri 网的不变式自动生成[130]和混成系统的不变式自动生成[131]. M. Colón 等人针对线性循环程序，预设线性不等式型模板，利用 Farkas 引理(参见文献[79, 134])生成非线性约束，通过求解约束得到预设模板形式的循环不变式[58]. 同时，斯坦福大学 Z. Manna 领

导的研究小组在文献[58, 132, 133]的基础上研发了不变式自动生成工具 StInG[2]，该工具仅针对线性程序生成等式型或不等式型线性循环不变式. 以上计算循环不变式的方法都是在忽略循环条件和分支条件的前提下工作的，陈应华等人首次提出了在考虑循环条件和分支条件的前提下，利用参系数半代数系统的实解分类求解循环不变式的算法[27, 54]. 该方法可以求解多项式等式型和不等式型循环不变式.

此外，E.Rodríguez-Carbonell 等人首次证明了多项式作为循环不变式具有理想(ideal)的代数结构[128]，结合抽象解释的观点[69]和 Gröbner 基的思想，提出了一个基于不动点计算的循环不变式自动生成算法[129]. 这种方法在理论上是完备的，可以生成一组多项式等式作为循环程序的不变式. 但是，该算法的时间复杂度是双指数的. L. Kovács 等人对一类特殊循环程序(P-Solvable)的不变式自动生成提出了一个完备的算法[100, 103, 104, 105, 106, 107, 108]. 该算法主要基于组合数学、多项式代数等理论基础，能够产生多项式等式型的不变式. 其所在研究小组还在 Mathematica 平台下开发设计了程序正确性验证系统 Theorema[9]，其中的软件包 Aligator 就是基于该算法来自动生成循环程序不变式. 这两种方法虽然在理论上都是完备的，但都只能处理某类特殊循环程序，而且这两种方法在生成循环不变式时依然忽略了循环条件和分支条件.

1.2.2 终止性分析研究概况

从 1.1 节的分析中可以看出，程序终止性分析对验证程序的完全正确性起着至关重要的作用. 程序终止性问题(program termination problem)，即众所周知的一致停机问题(uniform halting problem)，通常定义如下：

判断任意给定程序在有限时间内结束运行，或者该程序将永无休止地运行.

长期以来，停机问题一直为众多科学家们所关注. A. Turing 于 1936 年在文献[145]中证明了停机问题是不可判定的(undecidable). 后来，C. Strachey 于 1965 年在文献[139]中重申了这一观点. 然而，A. Turing 的这一理论结果使很多人产成了误解，他们认为：程序终止性证明是不可能实现的. 但恰恰相反，