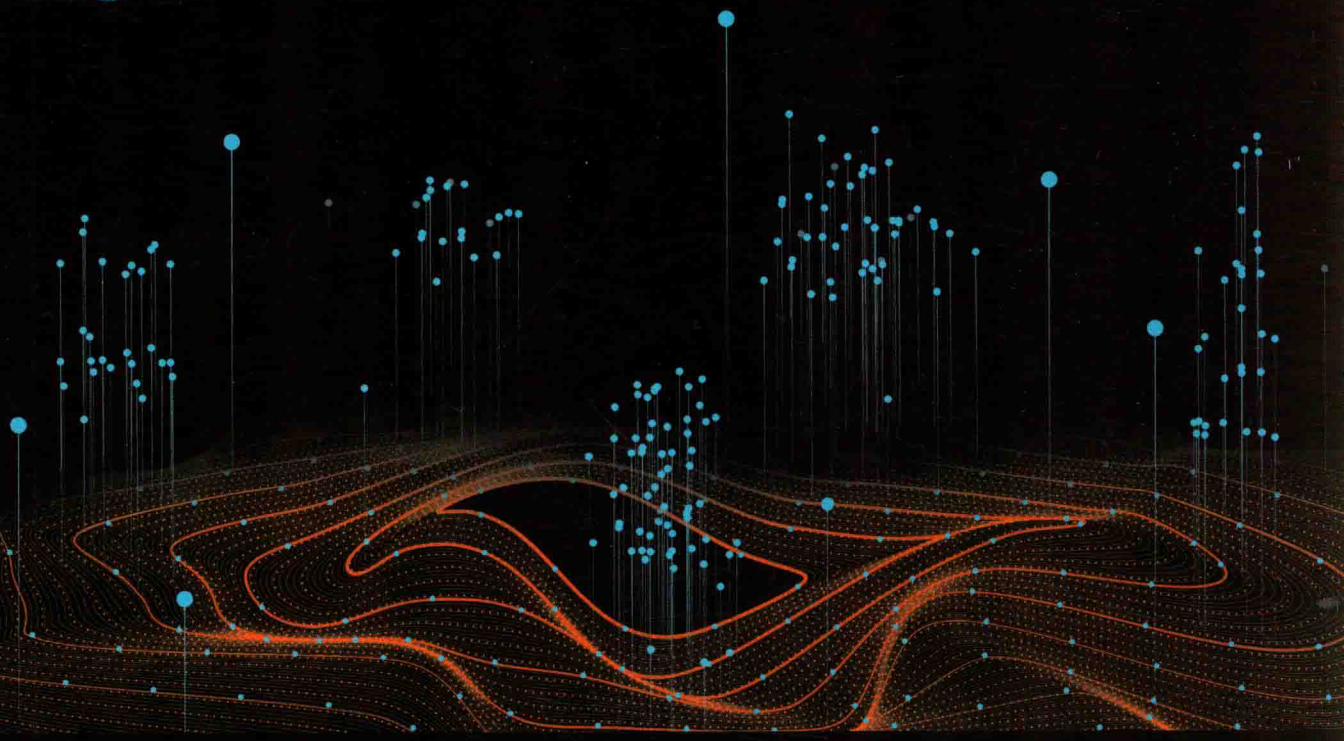
信息科学技术前沿丛书



基于程序分析的软件测试与错误定位技术

SOFTWARE TESTING AND FAULT LOCALIZATION TECHNOLOGY BASED ON PROGRAM ANALYSIS

易秋萍◎编著

北京邮电大学出版社
www.buptpress.com

信息科学技术前沿丛书

基于程序分析的软件测试与错误定位技术

易秋萍　编著

北京邮电大学出版社
www. buptpress. com

内 容 简 介

如今，软件在人们的社会生活中占据越来越重要的地位，软件的正确性也受到人们越来越多的重视。软件测试是保证软件正确性以及安全性的重要手段，它的主要任务是发现软件设计缺陷，进而要求开发人员分析、定位错误并修复缺陷。

本书共9章，其主要内容包括程序分析技术、符号执行技术、软件测试与错误定位技术、基于执行路径的最弱前置条件计算、基于后缀路径摘要的符号执行加速、基于反馈驱动的增量符号执行、级联式错误定位方法、演化软件错误定位方法和符号执行指导的并行程序分析。

本书是程序分析领域的专业书籍，可供软件测试、程序分析与验证领域的学生及研究人员学习和参考。

图书在版编目（CIP）数据

基于程序分析的软件测试与错误定位技术 / 易秋萍编著. -- 北京：北京邮电大学出版社，2023.8
ISBN 978-7-5635-6985-4

Ⅰ. ①基… Ⅱ. ①易… Ⅲ. ①软件—测试—错误校验—定位 Ⅳ. ①TP311.55

中国国家版本馆CIP数据核字（2023）第147596号

策划编辑：姚 顺 刘纳新 **责任编辑**：刘春棠 **责任校对**：张会良 **封面设计**：七星博纳

出版发行：北京邮电大学出版社
社 址：北京市海淀区西土城路10号
邮政编码：100876
发 行 部：电话：010-62282185 传真：010-62283578
E-mail：publish@bupt.edu.cn
经 销：各地新华书店
印 刷：北京虎彩文化传播有限公司
开 本：787 mm×1 092 mm 1/16
印 张：12.25
字 数：229千字
版 次：2023年8月第1版
印 次：2023年8月第1次印刷

ISBN 978-7-5635-6985-4 **定 价**：49.00元

前　言

软件广泛影响着人们生活的各个方面。计算机技术与各个领域的结合大大促进了这些领域的发展，而这些领域的发展也进一步加快了计算机技术的发展。随着人们对软件依赖程度的提高，人们期望日常使用的各类软件都是安全而可靠的。然而不幸的是，软件与程序的测试技术长期落后于软件开发技术。越来越多的程序种类、越来越大的代码规模以及越来越复杂的内部逻辑都使软件测试及维护变成了一项庞大的工程。据统计，软件测试及维护的成本占软件开发成本的50%～75%。软件测试是保证软件安全的重要手段，它的主要任务是发现软件设计缺陷，进而要求开发人员分析、定位错误并修复缺陷。

早期，软件测试主要由人工完成，测试人员通过对软件功能的理解构建测试输入与期望输出。后来，自动化软件测试技术出现并不断发展。随着计算机计算能力的提高以及约束求解等技术的不断发展，基于符号执行技术的测试用例自动构造方法得到了广泛的重视和应用。在测试阶段发现软件实现错误后，开发人员需要分析其产生错误的原因，进而修复该错误。但若修复方法不正确，则不但无法修复已发现的错误，相反还会带来其他缺陷甚至灾难。

自2008年以来，作者一直致力于程序分析测试以及验证方法和工具的研发，并在相关领域发表多篇CCF A类顶级会议/期刊论文，包括FSE、ICSE、TSE等。本书主要介绍作者从事程序分析技术研究以来，所提出的几种基于程序分析技术的软件分析测试以及错误定位方法。本书共9章。第1章概述程序分析技术，包括程序分析中常用的控制流分析以及数据流分析等。第2章介绍符号执行技术的基本思想以及原理，并分析讨论符号执行技术在发展过程中遇到的主要困难与挑战以及常见的符号执行工具，为后续章节的介绍做铺垫。第3章介绍软件测试与错误定位技术的研究背景，并概述经典的用于软件测试与错误定位分析的程序分析技术。第4章介绍基于执行路径的最弱前置条件计算框架，该框架是第5章、第7章以及第8章所述技术均使用的基本分析框架。第5章介绍一种有效的符号执行加速方法，该方法形式化描述执行路径间

共享的公共路径后缀，并通过避免它们的重复分析以有效缓解符号执行的路径爆炸问题。第 6 章介绍一种基于执行信息反馈驱动的增量符号执行技术，该方法总结已探索路径的行为，并保证在探索所有增量行为的前提下，在测试用例生成过程中裁剪与之前已探索路径覆盖相同增量行为的其余路径。第 7 章介绍一种级联式半自动化错误定位分析方法，它能系统化地产生导致错误发生的所有潜在原因，以帮助调试人员标识及理解真正的错误原因。所有错误原因被组织到一个树形结构中，便于调试人员分析理解错误与原因之间的因果关系。第 8 章介绍一种基于第 7 章所述方法的自动化演化软件错误定位方法，该方法是结合动态分析及语义分析的协同分析方法。它在标识引起错误发生的代码修改块集合的同时，产生有效信息解释所标识的代码修改块导致错误发生的原因。第 9 章介绍一种有效的并行程序验证方法 MPC，它是一种无状态模型检查技术。该方法同时分析程序的输入空间及线程调度空间，有效地减少在输入和调度空间之间存在的冗余分析。

限于作者水平，书中难免存在不足之处，敬请读者批评指正。

目　　录

第1章
程序分析技术

1.1 程序的正确性及其分析

软件正在改变我们的生活和工作方式。在生活中，我们通过智能手机上的社交软件来与朋友交流，通过网站购买所需要的东西，通过网络了解世界。在工作中，软件帮助我们组织业务、接触客户，帮助我们从众多竞争对手中脱颖而出。

然而，生产高质量的软件仍然面临着巨大的挑战，因此目前我们所使用的很多软件都依然存在各种类型的软件缺陷以及安全漏洞，如丰田汽车无法控制的加速问题、个人信息被剑桥分析公司从 Facebook 窃取的问题、Nest 智能恒温器出现故障致使许多家庭没有暖气的问题。仅仅是软件竞争这一种类型的软件缺陷在 2003 年就导致了一场大规模停电，并影响了美国的 8 个州和加拿大共 50 万人的生活。辐射治疗机器 Therac-25 中的软件缺陷使该机器设定了致命的剂量，在多次医疗事故中导致患者死亡或严重辐射灼伤。事后的调查发现整个软件系统没有经过严格的测试，而最初所做的 Therac-25 分析报告中有关系统安全的分析只考虑了系统硬件，没有把计算机故障包括软件故障所造成的隐患考虑在内。

如何保证程序的正确性是软件工程中的一个重要问题，也是一个一直都没有得到解决的问题。在软件的生产过程中，软件工程师是设计者，程序源代码是对软件设计的最精确的描述，可在机器上执行的二进制程序是最终的产品。从源代码的设计到可执行代码的建造过程是由编译器完成的。也就是说，软件从设计到产品的建造过程被自动化了。软件工程师只需要完成产品的设计，最终的产品就可以被自动建造出来。这样的自

动化过程尤其需要检验设计的正确性，以尽早发现可能存在的问题，减少重复劳动。

程序的源代码描述的是对内存状态的操作。一个输入状态经过程序处理后得到相应的输出状态。程序正确性则通过对程序运行过程中或者最终内存状态的描述来确定。对内存状态的正确性描述包括一些通用的规则，例如，被解引用的指针不能为空指针(NULL)，除数不能为0，等。非通用的正确性描述通常只适用于某个具体程序，需由人工显式地给出，例如，使用不同的性质描述语言。

在通常情况下，可以使用断言(assert 语句)来描述某一个程序点状态应满足的条件，这对程序员来说是非常方便的。尽管对程序在某个执行点处要满足的全部正确性条件进行描述是困难的，但是程序员在编写程序的时候，很容易知道程序要满足的部分正确性条件。某些条件是程序员假设默认成立的，但是这些条件往往不会在所有情况下都成立。这时如果程序员能够用断言将这些正确性条件添加到程序中，则对程序的正确性分析有较大的帮助。

在程序的执行过程中，程序在每个执行点的状态都是具体可知的。这时对断言或者其他正确性条件的检查就只需要读取并检查程序的内存状态是否满足相应的描述。这也正是现在应用广泛的软件动态测试方法的主要思想。基于动态执行的软件测试一次只能检查在一种输入下的正确性，而程序的输入通常是无穷多的，难以穷举所有可能的程序输入进行测试，从而很难保证程序的正确性。

程序分析是通过分析软件来了解其特性的有效技术。它可以发现上面提到的错误或者安全漏洞，还可以为软件合成测试用例，甚至可以自动给软件打补丁。例如，Facebook 使用 Getafix 工具为其他工具发现的漏洞自动生成补丁。另外，程序分析还可以用于编译优化阶段，以生成更高效的程序目标代码。

1.2 控制流分析

控制流分析的目的是根据程序中的跳转语句构造一个表达程序结构的控制流图，是数据流分析以及后端优化的基础。数据流分析可以在控制流图的基础上通过迭代分析得到人们感兴趣的数据流结果，如活性变量分析及可达定义分析等。

1.2.1 控制流图

程序分析工具通常将代码表示为控制流图(Control Flow Graph，CFG)。控制流图

是一个基于图形的程序控制流程的表示。它将简单的指令连接起来，静态捕获程序所有可能的执行路径，并定义程序中指令的执行顺序。当控制流依据不同的程序取值流向不同的方向时，相应地控制流图也有不同的分支。一个典型的例子是 if 语句或 while 语句的表示。在 if 指令的每个分支的指令集合后，合并不同分支并指向 if 分支结构后的指令。

控制流分析都是基于控制流图展开的。控制流图主要用于静态分析和编译应用程序，因为它可以准确地表示程序单元内部的控制以及数据流向。

程序中最简单的控制流单元是一个基本块——最大长度的无分支代码序列。它以带标签的操作开始，以分支、跳转或断言操作结束。基本块是一个操作序列，它们总是一起执行，除非某个操作引发异常。基本块只有一个入口和一个出口。入口是基本块的第一条语句，出口是基本块的最后一条语句。基本块的执行从其入口进入，从出口退出。

控制流图模拟程序中基本块之间的控制流。它是一个有向图，$G = (N, E)$。每个节点 $n \in N$ 对应一个基本块。每条边 $e = (n_i, n_j) \in E$ 对应可能从块 n_i 到块 n_j 的控制转移。控制流图包括每个基本块对应的节点，以及表示每个可能的基本块之间的控制转移的边。可以假设每个控制流图都有唯一的入口节点 entry 和唯一的出口节点 exit。对于一个程序的控制流图，entry 对应该程序的入口点。如果程序有多个入口，编译器可以插入一个唯一的 entry，并添加从 entry 到每个实际入口点的边。类似地，exit 对应程序的出口节点。而多个出口比多个入口更常见，编译器也可以很容易地添加一个唯一的 exit，并添加从每个实际出口到 exit 的边。为了表达的统一，可以为控制流图显式地添加一个入口节点 entry 和出口节点 exit。图 1-1(b)为图 1-1(a)中示例程序对应的控制流图。

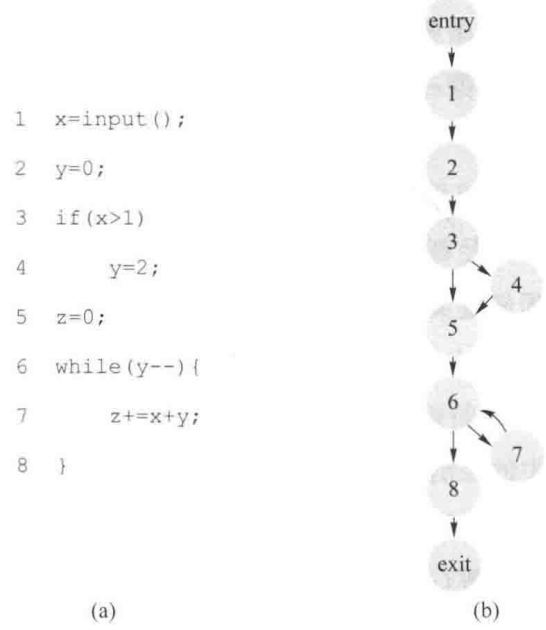


图 1-1 示例程序及其控制流图

对于流敏感分析，特别是数据流分析，语句顺序很重要，因此将程序看作控制流图会更方便。控制流图是程序代码的不同表示方式。这种程序的表示方法可以追溯到优化编译器的第一个程序分析器[1]。

1.2.2 程序依赖图

在控制流图中，如果每条从节点 entry 到节点 n 的路径都经过节点 d，则称节点 d 支配(dominate)节点 n，通常记为“d dom n”。因此，每个节点都支配它自己。支配树的根节点是控制流图的入口节点 entry，而每个节点 n 的父节点是它的直接支配节点，即任何路径上离 n 最近的支配节点。每个节点的直接支配节点是唯一的。

类似地，如果从节点 n 开始到节点 exit 的所有路径都经过节点 p，则称节点 p 后支配(post-dominate)节点 n，通常记为“p pdom n”。每个节点都后支配它自己。后支配树的根节点是控制流图的出口节点 exit，而每个节点 n 的父节点是它的直接后支配节点，即任何路径上 n 的第一个后支配节点。每个节点的直接后支配节点也是唯一的。

在顺序程序中，程序语句之间的依赖关系主要包括由控制条件和函数调用引起的控制依赖和由访问变量及参数传递引起的数据依赖。

(1) 控制依赖：在控制流图中，如果存在一条从节点 n 到节点 m 的路径 p，路径 p 上除 m 与 n 之外的其他节点 x 由 m 后支配，且 n 不由 m 后支配，则称节点 m 控制依赖节点 n。

(2) 数据依赖：在控制流图中，如果变量 v 在节点 n 被定义，在节点 m 处被使用，且存在一条从节点 n 到 m 的路径，变量 v 在此路径上除节点 n 外未被重新定义，则称节点 m 数据依赖节点 n。

图 1-2 描述了一个控制流图和对应的支配树及后支配树。节点 4 支配节点 5，因为从节点 1 到节点 5 的所有路径都必须经过节点 4；节点 7 后支配节点 4，因为从节点 4 到节点 9 的所有路径都必须经过节点 7。此外，节点 6 控制依赖节点 4，但是节点 7 并不控制依赖节点 4。

给定的程序语句 m 和语句 n 可能通过控制流或者数据流而联系在一起。控制流图主要用来描述一个程序中的控制流，如果在控制流中添加所有的控制依赖和数据依赖关系，则构成相应的程序依赖图(Program Dependence Graph，PDG)。

程序依赖图是一个有向图 $G=(N,E)$，其中 N 为程序语句对应的节点集合，E 为边集合。边集合表示程序中基本的控制依赖关系和数据依赖关系。程序依赖图只能用于

描述单一函数的依赖信息。为了表达由多函数组成的实际程序中的依赖关系，需将程序依赖图进一步扩展为可以描述复杂程序(由多个函数构成)的系统依赖图。

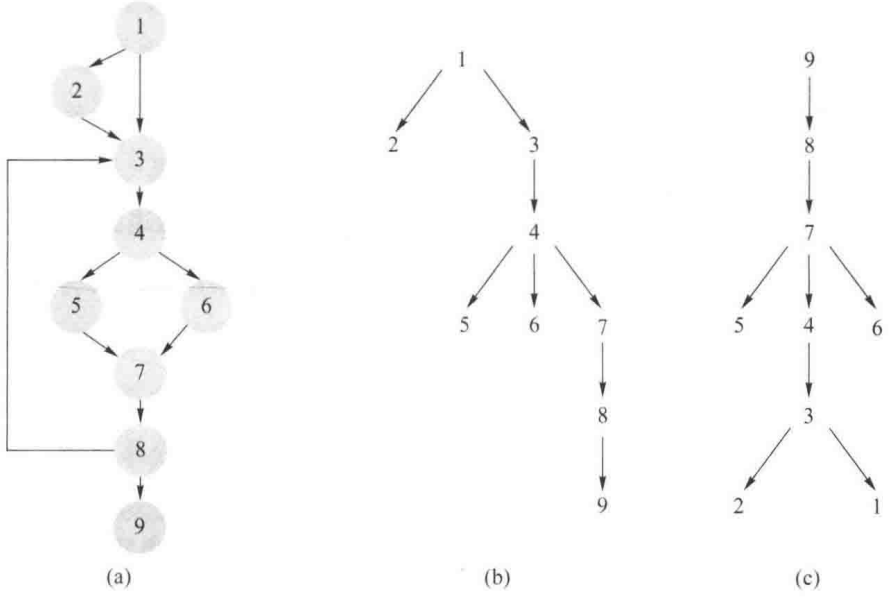


图 1-2 控制流图及其支配树与后支配树

1.2.3 系统依赖图

系统依赖图(System Dependence Graph，SDG)是程序依赖图的扩展，用来描述多个函数相互作用而构建的复杂程序的依赖关系。它是在每个函数的程序依赖图上增加一些额外的点和边将整个系统整合在一起而来的。系统依赖图是多个程序间的数据流和控制流信息的有效表达方式，是精确过程间分析的基础。

在程序依赖图上，系统依赖图增加了 5 类新的节点。

(1) 函数调用节点：描述被调用函数的信息。

(2) actual-in 节点：有与调用点相关的控制依赖，将实参的值传入一个临时单元中。

(3) actual-out 节点：有与调用点相关的控制依赖，将临时单元的值返回给实参。

(4) formal-in 节点：有与被调函数入口相关的控制依赖，将临时单元的值复制给形参。

(5) formal-out 节点：有与被调函数入口相关的控制依赖，将形参的值返回给临时单元。

每个调用点在系统依赖图中都有对应的函数调用节点、actual-in 节点和 actual-out

节点。每个程序依赖图都有一个 entry 节点、formal-in 节点和 formal-out 节点。actual-in 节点和 actual-out 节点控制依赖函数调用点，formal-in 节点和 formal-out 节点控制依赖 entry 节点。

系统依赖图中增加了 3 类新的边。

(1) 从调用点指向被调函数入口节点的边：一种新的控制依赖边。

(2) parameter-in 边：actual-in 节点指向 formal-in 节点的边。

(3) parameter-out 边：formal-out 节点指向 actual-out 节点的边。

parameter-in 边和 parameter-out 边是新的数据依赖边。

1.3 数据流分析

数据流分析是一种编译时使用的技术，它能从程序代码中收集程序的语义信息，并通过代数的方法在编译时确定变量的定义和使用。数据流分析不必实际运行程序就能够发现程序运行时的行为，进而可以帮助用户理解程序。数据流分析被用于解决编译优化、程序验证、调试、测试、并行、向量化等问题。数据流分析的目的是解释程序如何操作数据信息，其应用范围非常广泛。

数据流分析的计算非常复杂，尤其是过程间的数据流分析，因为过程间的调用关系比较复杂，使得静态的数据流分析更为困难。然而在所有的程序点计算完整的数据流信息并不总是必要的。

1.3.1 数据流分析概述

经典的数据流分析从控制流图和具有有限高度的完整格开始。格描述了我们希望推断出的不同控制流图节点的抽象信息。数据流分析在控制流图中的每个程序点计算一些数据流信息(假设用 σ 来表示)。在通常情况下，σ 用于描述程序中每个变量的一些信息。例如，σ 可以将变量映射到某个集合 L 中的抽象值，其中 L 表示在分析中感兴趣的一组抽象值，它在不同的分析中通常不尽相同。例如，对于零分析(zero analysis)，它跟踪每个变量看是否每个变量在每个程序点上都为零或不为零。对于零分析，我们可以定义 L 为集合$\{Z,N,T\}$。其中，抽象值 Z 表示值 0，N 表示所有的非零值，T 表示所有不精确分析带来的不确定情况。