并发程序变异测试研究综述

田 甜[1],巩敦卫[2]
(1. 山东建筑大学计算机科学与技术学院,山东济南 250101; 2. 中国矿业大学信息与控制工程学院,江苏徐州 221116)

摘 要: 变异测试是一种面向缺陷的软件测试方法,利用人为注入的缺陷引导测试数据生成,评价测试数据的有效性,在软件工程领域得到了广泛关注. 依托多核架构,开发可靠的并发程序越来越迫切. 近年来,很多学者尝试将变异测试技术应用于并发程序,以提高并发程序测试的效率和可靠性. 首先,介绍了本文工作与已有综述的不同;然后,阐述了与并发程序和变异测试技术相关的知识;接着,从变异实施、变异测试准则、测试数据生成等 3 方面,综述并发程序变异测试的研究进展,包括:变异算子设计、选择变异、高阶变异、弱变异、测试数据生成方法、变异测试工具等;最后,讨论需要进一步研究的问题.

关键词: 并发程序; 变异测试; 变异算子; 优化; 测试数据生成
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112(2020)11-2267-11
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2020.11.025

Survey on Mutation Testing of Concurrent Programs

TIAN Tian[1], GONG Dun-wei[2]
(1. *School of Computer Science and Technology, Shandong Jianzhu University, Jinan, Shandong* 250101, *China*;
2. *School of Information and Control Engineering, China University of Mining and Technology, Xuzhou, Jiangsu* 221116, *China*)

Abstract: Mutation testing is a fault-oriented software testing method, which adopts manually injected defects for guiding the generation of test data and evaluating their adequacy. Researchers in the community of software engineering have paid much attention to mutation testing. The prevalence of multi-core architecture makes an ever increasing need for developing reliable concurrent programs. Many scholars have attempted to employ mutation testing to concurrent programs, with the purpose of improving the reliability and efficiency of testing these programs in recent years. Firstly, the difference between this study and previous reviews is displayed. Then, following the background related to concurrent programs and mutation testing, this study surveys the progress on mutation testing of concurrent programs from the following three aspects, mutation implementation, mutation testing adequacy criteria, and test data generation. Specifically, it surveys techniques related to optimizing mutation testing from such aspects as designing mutation operators, selective mutation, high-order mutation, weak mutation, methods of test data generation, and prototype tools. Finally, this study discusses some topics to be further researched.

Key words: concurrent program; mutation testing; mutation operator; optimization; test data generation

1 引言

并发程序是指包含多个并发执行流程的程序. 这些执行流程可以同时执行,并在执行过程中相互通信. 多核架构和多核并行计算技术的快速发展,以及日益增长的计算需求和人们对计算速度要求的提高,使得并发程序越来越普遍. 迄今为止,并发程序已经在图像处理、生物医药、电力系统等多个领域得到了广泛应用. 测试是保证并发程序质量、提高并发程序可靠性的重要手段[1]. 多个执行流程之间的交互、程序运行的不确定性使得并发缺陷难以检测和修复[2],给并发程序的测试带来很大难度. 近年来,学者们在并发程序测试、缺陷发现和检测等领域取得了很多成果[2,3].

总体来讲,并发程序测试有 3 种方法,分别为:静态分析法、动态分析法,以及模型检测法. 静态分析法,顾名思义,静态分析被测程序代码,不用实际运行程序,从

收稿日期:2019-10-29;修回日期:2020-01-27;责任编辑:梅志强
基金项目:国家自然科学基金(No. 61773384, No. 61503220);国家重点研发计划(No. 2018YFB1003800-02-01)

而发现可能引起错误的情景[4];动态分析法则需要实际运行被测对象,并监控运行过程[5];作为一类重量级的静态方法,模型检测的基本思想是:基于被测程序模型,搜索程序可能的行为空间,从而判断程序是否存在缺陷.然而,当并发程序规模增大时,容易出现空间爆炸问题,导致此类方法的扩展性较差[6].进一步,混合法综合了静态和动态方法的优点,并在一定程度上克服了它们的不足[7].

变异测试的概念最早起源于1971年[8],DeMillo等在1978年的工作被认为是变异测试领域的开创性研究[9,10].作为一种基于缺陷的软件分析技术,变异测试通过对被测程序注入特定类型的缺陷,评估测试数据的缺陷检测能力,基本思想如下:首先,基于一定的规则,对被测程序实施微小的修改,产生一个仍然满足语法要求的新程序,称这个新程序为变异体;然后,给定同样的测试数据,运行被测程序和变异体,如果它们的最终输出结果相同,说明变异没有被发现,否则,认为测试数据检测到了变异[9].

自1970年以来,变异测试技术取得了蓬勃发展,不仅用于串行程序测试,也用于并发程序[11]、机器学习程序[12]、事件驱动的Web程序[13]等的测试;除了作为测试数据生成[14]和测试充分性验证[15]的有效手段之外,还用于软件缺陷预测[16].在并发程序测试方面,并发程序变异测试的研究主要集中在变异体生成和变异测试优化等2个方面,研究成果涵盖并发变异算子设计、变异体约减、变异体选择,以及变异测试数据生成等.

Jia等从计算开销约减、等价变异体检测、变异测试应用、实验评价,以及变异测试工具等方面,综述了变异测试的研究成果[17].陈翔等从原理分析、测试数据生成,以及回归测试等角度,对Jia等的工作进行了补充[18].Offutt回顾了变异测试的发展历程,阐述了当时的研究状况,预期了未来的研究动向,从变异算子的理论研究、变异测试与软件开发过程的集成等方面,提出未来可能的研究切入点[10].Madeyski等综述了解决等价变异体问题的研究成果[19].Souza等和Silva等系统回顾了基于变异的测试数据技术[20]和搜索方法在变异测试的应用[21].Belli等综述了基于模型的变异测试进展,即变异测试在有向图、事件序列图,以及有限状态机和状态图的应用[22].Nguyen等分析了变异测试在变异体数量、揭露真实缺陷,以及等价变异体等3个方面的局限性,讨论了高阶变异解决这些问题的优势和存在的问题[23].苏等讨论并发缺陷之间的关系,从缺陷暴露、检测和规避等方面,对已有工作进行了综述[2].Bianchi等回顾了已有并发测试技术,并进行分类和对比,分析每种方法的优缺点[3].Melo等从实证研究的角度,对并发程序测试的相关工作进行了总结和分类[24].Pizzoleto等综述了降低变异测试代价的技术和标准[25].Kintis等对比了3个Java变异测试工具MUJAVA、PIT和MAJOR的缺陷检测能力,使用的变异算子主要针对算术、逻辑和关系运算等,并没有针对并发特性和并发程序变异测试进行讨论[26].

这些工作分别从不同侧面回顾了变异测试和并发程序测试的研究成果,为研究成果的推广和相关测试技术的进一步探索提供了帮助.然而,上述研究尚缺乏从并发程序的角度,针对变异测试技术的分析和比较.Arora等从覆盖测试、基于模型的测试,以及变异测试等多个角度,综述了并发程序的研究成果.在变异测试方面,重在面向缺陷的测试技术在并发程序的应用,以及变异算子设计和变异体执行的相关工作[11].与上述工作不同,本文从并发程序的并发、通信,以及交互等特点出发,分别针对不同类型的并发程序,阐述变异体产生、约减和选择,充分性准则,以及测试数据生成等方面的研究成果,并讨论需要解决的问题,进而展望可能的研究方向.

2 并发程序与变异测试

并发程序包含多个执行流程.假设程序P包含m个流程$e_1,e_2,\cdots,e_m$,这些流程并发执行、相互通信,记作$P=\{e_1,e_2,\cdots,e_m\}$.P可以接受的输入包含n个分量:$i_1,i_2,\cdots,i_n$,记为$\boldsymbol{I}=(i_1,i_2,\cdots,i_n)$.给定一个程序输入,$P$能够产生相应的输出结果.

基于程序不同的存储机制,并发程序分为分布存储和共享存储两种形式.顾名思义,分布存储并发程序的每个执行流程拥有独立的存储器,不存在共享存储器;共享存储并发程序的所有执行流程除了具有独立存储器之外,还有共享存储单元,多个执行流程基于不同的存储形式,使用一定的消息传递方式交互[3].并发、交互、同步是并发程序的典型特征,其中,同步机制用于避免流程冲突和存储一致性错误,以及消息通信产生的问题.

图1展示一个生产者和消费者并发程序[27],记为$P=\{e_1,e_2\}$,S为共享存储器,规模为2,e_1为生产者流程,负责读入字符,并存放到共享存储器S中;e_2为消费者流程,负责从存储器中取出字符并输出.当存储器有2个字符时,生产者流程不能再向存储器存入字符;e_1和

e_1	e_2
get(S)	Withdraw(S)
Deposit(S)	put(S)
get(S)	Withdraw (S)
Deposit(S)	put(S)
get(S)	Withdraw (S)
Deposit(S)	put (S)

图1 示例并行程序

e_2并发执行,通过对 S 的存取交互.

给定被测程序 P,对 P 实施变异,也就是说,按照一定规则对 P 作微小的改动,得到区别于 P 的新程序 P',称 P'为变异体. 对 P 实施变异时,依据的修改规则叫作变异算子. 若只对程序 P 使用一个变异算子实施一次变异,称得到的新程序为一阶变异体. 相应地,若对 P 实施 $m(m>1)$ 次变异,形成的变异体称为高价变异体[28]. 图2展示了一个典型的变异实施操作. 对P的赋值语句"$a=a+10$"实施变异操作,将操作符"+"改变为"-"生成变异体 P',对应的变异算子属于算数运算符替代 AOR(Arithmetic Operator Replacement).

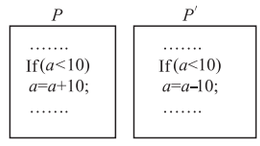

图2　变异操作

表 1　串行程序的变异算子

变异算子	作用
AAR(Array Reference for Array Reference Replacement)	更换数组引用
ABS(Absolute Value Insertion)	绝对值插入
ACR(Array Reference for Constant Replacement)	常量被数组引用替换
AOR(Arithmetic Operator Replacement)	算术运算符更换
ASR(Array Reference for Scalar Variable Replacement)	变量被数组引用替换
CAR(Constant for Array Reference Replacement)	数组引用被常量替换
CNR(Comparable Array Name Replacement)	更换数组名
CRP(Constant Replacement)	更换常量
CSR(Constantfor Scalar Variable Replacement)	变量被常量替换
DER(DO Statement Alterations)	修改 Do 语句
DSA(DATA Statement Alterations)	修改 Data 语句
GLR(GOTO Label Replacement)	更换 Goto 标签
LCR(Logical Connector Replacement)	更换逻辑连接符
ROR(Relational Operator Replacement)	更换关系运算符
RSR(RETURN Statement Replacement)	更换 RETURN 语句
SAN(Statement Analysis)	语句分析
SAR(Scalar Variable for Array Reference Replacement)	数组引用被变量替换
SCR(Scalar for Constant Replacement)	常量被变量替换
SDL(Statement Deletion)	删除语句
SRC(Source Constant Replacement)	更换源常量
SVR(Scalar Variable Replacement)	更换变量
UOI(Unary Operator Insertion)	一元操作符插入

使用某一测试数据运行 P 及其变异体,如果两个程序的输出结果不同,那么说明变异体被发现,或者称变异体被杀死. 如果 P'与 P 的语义相同,也就是说,在所有输入下,P'与 P 的输出都相同,则称 P'为等价变异体. 如果 P 有 m 个变异体,其中,等价变异体数为 m',某测试数据集杀死的变异体数为 k,那么,该测试数据集的变异得分为:$k/(m-m')$. 测试数据的有效性正是通过变异得分衡量的.

变异测试最早应用于串行程序. King 等针对 Fortran77 提出包含 AOR 在内的 22 个变异算子,全面考虑了串行执行流程,如表 1 所列[29]. 为其它类型程序变异算子的设计和后续变异测试的拓展研究打下了基础. 伴随在串行程序测试的发展,变异测试也应用于各种类型的并发程序中.

当变异测试技术用于并发程序时,主要有以下 2 点与串行程序不同:(1)适用于串行程序的变异算子体现被测程序的计算错误、逻辑错误和控制错误等,覆盖程序的串行执行流程;并发程序变异测试除了考虑这些因素之外,还要设计与并发执行相关的变异算子,以模拟多流程并发和交互缺陷;(2)由于不同流程的执行顺序不同,对并发程序而言,在同一测试数据下,原程序和变异体一次相同(不同)的运行结果并不说明该测

试数据不能(能够)杀死变异体.

3 并发程序变异测试

3.1 变异实施

变异算子是变异测试首先需要考虑的因素. 考虑与并发执行相关的同步设施[2]、资源分配、数据传递,以及参数设置等因素,能够设计与并发相关的变异算子.

Ada 语言本身提供了对并发程序设计的支持,使用任务作为并发模块,并提供了保障任务之间通信的汇合机制[30]. Offutt 等设计了 65 个用于 Ada 语言的变异算子[31],其中,任务算子有入口语句修改、接受语句替换,以及选择语句替换等,涉及并发任务的执行. 为了反映 Java 并发程序的可测试性,Ghosh 定义了与移除同步关键字相关的两个变异算子[32]. Delamaro 等针对 Java 语言的并发和同步等特点,对程序负责同步的代码,提出能够反应程序并发缺陷的变异算子集合,包括:删除同步属性、删除对同步方法的调用、替换同步对象、删除和替换与并发相关的函数,以及转换对同步方法的调用参数等 11 个算子[33]. Bradbury 等提出了更充分的并发变异算子,如表 2 所列,包括:修改并发方法的参数和发生、修改关键字、交换并发对象,以及修改关键域等,并将这些变异算子与并发缺陷模式对应[34]. 该文被 Java 并发程序变异测试的相关研究广泛引用. 此外,针对 Java 8 新引进的并发特性和标准,Wu 等分析并发处理的关键因素,设计并优化了新的变异算子集合[35].

表 2 Java 程序并发变异算子

分类	变异算子	描述
修改并发方法参数	MXT(Modify Method-X Time)	修改 wait(),sleep(),join(),await()方法的时间参数
	MSP(Modify Synchronized Block Parameter)	修改同步块参数
	ESP(Exchange Synchronized Block Parameters)	交换同步块参数
	MSF(Modify Semaphore Fairness)	修改信号公平设置
	MXC(Modify Permit Count in Semaphore and Modify Thread Count in Latches and Barriers)	修改并发机制的信号量或线程数
	MBR(Modify Barrier Runnable Parameter)	修改同步障参数
修改并发方法调用的发生	RTXC(Remove Thread Method-X Call)	移除 wait(),join(),sleep(),yield(),notify(),notifyall()方法的调用
	RCXC(Remove Concurrency Mechanism Method-X Call)	移除并发机制方法的调用
	RNA(ReplaceNotifyAll()with Notify)	用 Notfiy()替换 NotifyAll()
	RJS(Replace Join()with Sleep())	用 Sleep()替换 Join()
	ELPA(Exchange Lock/Permit Acquisition)	改变共享资源的申请方法
	SAN(Exchange Atomic Call with Non-Atomic)	改变原子调用为非原子调用
修改关键字	ASTK(Add Static Keyword to Method)	为方法增加静态关键字
	RSTK(Remove Static Keyword from Method)	移除方法的静态关键字
	ASK(Add Synchronized Keyword to Method)	为非同步方法增加同步关键字
	RSK(Remove Synchronized Keyword from Method)	移走方法的同步关键字
	RSB(Remove Synchronized Block)	移走同步块
	RVK(Remove Volatile Keyword)	移走关键字 Volatile
	RFU(Remove Finally Around Unlock)	移除 Unlock 附近的 Finally 关键字
改变并发对象	RXO(Replace One Concurrency Mechanism-X with Another)	替换同类并发机制
	EELO(Exchange Explicit Lock Objects)	交换显示锁对象
修改关键域	SHCR(Shift Critical Region)	转移临界区域
	SKCR(Shrink Critical Region)	缩减临界区域
	EXCR(Expand Critical Region)	扩展临界区域
	SPCR(Split Critical Region)	分解临界区域

为了消除 System C[36]程序的潜在缺陷,Sen 使用变异测试对系统验证,提出 7 个用于 System C 的变异算子,分为 2 类:修改与并发操作相关的参数,以及移除、替换和改变并发操作[37]. Nilsson 等针对实时系统任务的并发和不确定性,考虑任务的执行时间、共享资源的使用,以及环境行为的改变等,提出了变异算子[38]. Actor 程序基于消息传递实现并发线程的同步[39]. Jagannath 等提出了用于测试 Actor 程序的 12 个变异算子,分为消息、消息约束,以及线程创建等 3 类[40]. MPI 是常用的分布存储并行程序开发消息传递环境[41],针对 MPI 通信函数,Silva 等提出了 26 个变异算子,按照适用范围分为 3 类:用于集合通信、用于点到点通信,以及全部 MPI 函数[42]. 这些变异算子和 Actor 程序的变异算子有异曲同工之处,主要对通信和并发函数,以及相应的参数修改. Cañizares 等针对仿真的云和高性能计算环境,设计了变异算子,复现分布式系统的通信和死锁错误[43]. Estero-Botaro 等针对 WS-BPEL 业务流程,设计了与并发控制流相关的变异算子[44]. 此外,考虑对描述语言本身的误解导致的软件错误,Clark 等提出了语义变异的概念[45]. 所谓语义变异,是指直接改变编程语言的语义生成变异体. Cao 等研究了并发程序的语义变异,用于测试共享存储程序[46]. 但是,只在实验部分提出变异算子,尚缺乏充分的并发语义变异算子设计.

昂贵的计算代价一直是变异测试在实际工业程序应用的障碍. 为了提高变异测试的效率,约减变异体是最直接和合理的途径[47]. 主流的策略有:随机变异、选择变异、高阶变异,以及基于变异体位置的方法等[48]. 随机变异是从变异体集合中,随机选择一定比例的变异体[49]. 考虑变异算子的重要性,选择变异从变异算子集合中选择一个子集,基于该子集生成变异体,使得杀死后者的测试数据集能够杀死全部变异体[50]. Gligoric 等首次研究了 Java 并发程序的选择变异,在表 2 的基础上增加 3 个新的变异算子,并通过实验给出了多个变异算子集合,为平衡并发变异测试代价和效率提供了多种选择[51].

考虑 Bradbury 等[34]和 Delamaro 等[33]提出的与同步方法相关的变异算子,Wu 等还提出使用以同步中心的二阶变异算子集合,以体现一阶变异不能表示的并发缺陷[52]. Kusano 等提出局部变异的概念,主要思想是只对重要的方法或复杂的程序部分实施变异[53]. 开发的工具 CCmutator 也是首个多线程 C/C ++ 程序生成并发变异体的工具,它应用了并发机制,能够高效并行的产生同阶变异体. 该工具只针对变异体产生,没有提供变异执行,以及结果判定等.

类似地,Madiraju 等针对 Java 并发程序的复杂模块实施一阶和高阶变异,以降低变异测试的代价[54]. 虽然开发的变异测试工具 Paraμ 包含变异体产生和分析,以及测试数据集的有效性评价等,但是,仅针对部分变异算子实现变异体生成,完整的工具功能还没有完成. 针对 WS-BPEL 业务流程,孙等通过高阶变异和变异算子优先级排序两种途径,降低变异体数量,并开发了变异测试工具 μBPEL[55]. 表 3 列出了不同类型并发程序变异实施的工作以及对应的文献.

表 3　变异实施

程序	相关工作	文献
Java 程序	变异算子设计	Bradbury 等[34]
	变异算子设计	Ghosh[32]
	变异算子设计	Delamaro 等[33]
	变异算子设计	Cao 等[46]
	选择变异	Gligoric 等[51]
	高阶变异/部分变异	Madiraju 等[54]
Ada 程序	变异算子设计	Offutt 等[31]
SystemC 程序	变异算子设计	Sen 等[37]
实时系统	变异算子设计	Nillson 等[38]
Actor 程序	变异算子设计	Jagannath 等[40]
MPI 程序	变异算子设计	Silva 等[42]
Java 程序/Erlang 程序	高阶变异	Wu 等[52]
C/C ++ 程序	高阶变异	Kusano 等[53]
模拟分布式系统	变异算子设计	Cañizares 等[43]
WS-BPEL 业务流程	变异算子设计	Estero-Botaro 等[44]
	高阶变异/ 变异算子优先级	孙等[55]

3.2　变异测试充分性准则

以程序最终输出是否相同判断变异体是否被杀死,是传统的变异测试准则[17],也称强变异测试准则,可达性、必要性和充分性是杀死变异体的测试数据需要满足的 3 个条件[56],其中,可达性指测试数据能够到达变异出现的位置;必要性指测试数据能够使程序在变异位置的状态发生改变;充分性意味着状态改变能导致程序输出不同.

对串行程序而言,判断一个测试数据是否杀死变异体,只需在测试数据下运行原程序及其变异体,判断两个程序的输出是否相同. 然而,如前所述,并发程序和变异体分别在测试数据下的一次输出相同,不能说明该测试数据杀不死变异体,类似的,输出结果不同,也不能说明变异体被杀死. 这是因为,并发程序的执行,不仅与程序输入有关,还与流程交互顺序有关. 相同的程序输入,不同的流程交互顺序可能导致不同的程序输出结果. 因此,对并发程序而言,判断变异体能否被某一测试数据杀死,不再是比较变异体和原程序

的输出是否相同,而是需要比较变异体和原程序的两个输出集合是否相同,这是并发程序变异测试执行代价高的关键原因.

与强变异对应的另一判定标准是弱变异测试准则[57]. 弱变异的概念由 Howden 等在 1982 年提出,基本原理如下:假设 P 由多个元素组成,对 P 的某一元素实施变异得到变异体 P',基于某一测试数据运行 P 和 P';如果程序 P 在该元素处的执行与 P'不同,则称该测试数据杀死变异体. 可以看出,弱变异和强变异测试准则的区别在于,变异体是否被杀死的依据不再是原程序和变异体的输出是否不同,而是二者在变异元素处的中间状态是否不同. 也就是说,测试数据仅需要考虑可达和必要性,充分性不再是必须满足的条件.

弱变异测试准则下,测试数据只需满足可达和必要性条件,即可判断变异体是否被杀死,而不需要执行整个程序,使得执行代价低于强变异测试[58]. 虽然弱变异测试摒除了充分性,不能保证变异位置的状态改变传播到程序输出,但是,已有关于串行程序的理论和实验表明,基于特定条件,弱变异能够生成与强变异相同效力的测试数据[59]. Papadakis 等考虑一阶弱变异测试,利用变异测试的必要性条件,融合原语句和变异后语句构建相应的条件语句,将这些条件语句依序插入原程序中,得到一个新程序,进而将变异测试数据生成转变为新程序中相应条件语句的真分支覆盖问题[60]. 受此启发,我们将弱变异测试应用于 MPI 消息传递并发程序中,针对通信语句的特点,以及与通信语句相关的不同进程,提出变异相关条件语句的构建方法,并在原程序的合适位置插入条件语句,完成了并发程序弱变异测试数据生成问题的转化[61]. 该研究没有考虑并发变异算子之间的关系. 如果构建条件语句时选择部分有代表性的变异算子,那么,将能进一步提高弱变异测试的效率.

固定变异测试是介于强变异和弱变异之间的一种中间策略,适用于交互式开发环境,能够独立运行部分代码片段[62]. 与其它两种方法相比,固定变异测试考虑程序的多个元素,而不仅仅是弱变异测试准则着重的一个元素;可以只运行部分程序,不需要执行强变异准则要求的整个程序;能够综合考虑被测程序和变异代码段在各阶段的运行结果[63]. 假设对 P 的某一元素实施变异,得到变异体 P',那么固定变异测试的核心在于,比较源程序和变异体从该元素到程序结束之间的状态. 因此,强变异和弱变异准则可以看作固定变异策略的两个特例. 具体地讲,当在变异元素处比较程序状态时,固定变异对应弱变异策略;如果在程序运行结束时比较源程序和变异体的运行结果,那么,固定变异即为强变异策略[64]. Estero-Botaro 等将固定变异策略应用到 WS-BPEL 业务流程测试中[65].

3.3 变异测试数据生成

实施变异测试的关键步骤之一是基于测试数据运行原程序和变异体,以判定变异体是否能够被杀死. 鉴于并发程序测试不仅与测试数据有关,还与流程交互顺序有关,本节不仅阐述测试数据生成的研究成果,还有流程交互顺序选择的相关工作.

Carver 首次将变异测试应用于并发程序,使用 King 等设计的变异算子[29]对 Ada 并发程序变异[27]. 考虑并发程序不确定性导致的问题,提出了确定执行变异测试. 基于多次运行产生流程交互顺序,使用输入和流程交互顺序作为测试数据,提高了变异测试的效率. Gligoric 等执行原程序时收集必要的信息,减少变异体执行的状态搜索,开发了变异测试工具 MuTMuT,主要思想是:将变异测试分为两个阶段,第一阶段执行原程序,收集相关信息;第二阶段利用第一阶段收集的信息,减少状态搜索空间,加快寻找杀死变异体的流程交互顺序的速度[66]. 但是,MuTMuT 在前期收集可用信息时,需要的计算和存储代价较大. 此外,虽然能够处理并发变异,但无法使用并发变异算子自动生成变异体. 使用类似的优化技术,针对并发变异算子,Gligoric 等在实施变异测试时,判断原程序基于测试数据能否执行到变异发生的位置,从而决定能否执行相应的变异体. 开发的变异测试工具 Comutation 能够生成和执行变异体,以及判定变异体能否被杀死[51]. 同样的思想也用于 System C 并发程序的测试中[67].

随机测试是一种最基本的软件测试策略,在工业和学术界得到广泛应用. 在变异测试领域,随机测试的主要思想是:随机生成测试数据,验证生成的数据是否能够杀死变异体. 例如,Cañizares 等通过对分布式环境模拟,规避分布式程序的不确定性执行,并使用随机方法生变异测试数据,开发了工具 MuTomVo,以测试包含 API 或外部库调用的应用程序[43].

虽然这些方法缓解了不确定性对并发程序变异测试的影响,降低了程序或变异体的执行代价,但并没有充分利用程序知识,无法引导性的生成有效的测试输入或线程交互顺序,称这些方法为传统方法. 与之对应的是基于启发式方法的变异测试数据生成.

很多学者将启发式方法应用于解决软件工程的一些难题,形成了热门研究领域-基于搜索的软件工程[68,69],其中,研究最为广泛的是基于搜索的测试数据生成[70,71]. Silva 使用排列的方式,将 Java 程序的流程交互顺序集合编码为进化个体,使用变异得分评价进化个体的优劣,还设计了符合个体编码形式的交叉和变异操作,基于生成的流程交互顺序集合,测试数据能够达到很高的变异得分[72]. Cao 等组合符号执行和进化

方法生成测试数据,首先,使用符号执行方法生成弱变异测试数据;然后,使用进化方法搜索从变异点到输出点的最优路径,使生成的测试用例符合强变异测试准则[46]. Nilsson 等针对实时系统的多任务并发特性,利用遗传算法生成变异测试数据,检测变异体的潜在缺陷[73]. Ghiduk 等使用变异得分作为适应度函数,生成杀死高阶并发变异算子的测试数据[74]. Palomo-Lozano 等还使用整数线性规划,寻找具有最小执行代价的变异测试数据集[64].

此外,Takagi 等将变异测试引入基于模型的测试中,将并发程序建模为位置/转换网,对库所/变迁网实施变异,提出了包含变异体生成、测试用例生成,以及变异分析器的测试框架[75],但没有给出具体的实施方法. 此外,针对实时并发软件,Nilsson 等借助模型检查器,生成变异测试数据[38],为利用模型检测工具有效生成变异测试数据提供了思路. 表 4 列出了不同测试数据生成方法、目标程序和对应文献.

表 4　测试数据生成方法

方法	程序	文献
一般方法	Ada 程序	Carver[27]
	Java 程序	Gligoric 等[66]
	Java 程序	Gligoric 等[51]
	SystemC 程序	Sen 等[67]
	模拟分布式系统	Cañizares 等[43]
基于搜索的方法	Java 程序	Silva[72]
		Ghiduk 等[74]
	实时系统	Nilsson 等[73]
	WS-BPEL 业务流程	Palomo-Lozano 等[64]
基于模型的方法	并行系统	Takagi 等[75]
	实时系统	Nilsson 等[38]

4　需要进一步研究的问题

可以看出,与变异测试在串行程序应用取得的大量研究成果相比,并发程序的变异测试研究成果相对较少. 主要原因在于,并发、交互和不确定等特性使得并发程序的变异测试难度增大;另外,并发程序的实现途径多,通常涉及具体的并发环境和技术,导致对测试研究人员的背景知识要求高,也是并发程序变异测试研究成果相对较少的因素之一. 结合并发程序的特点,充分利用变异测试的优势,更有效的应用变异测试技术到并发程序中是非常必要的.

(1)为了提高变异体产生的效率,选择部分有代表性的变异算子,从而减少变异体数量是非常有必要的. 学者们已经提出了选择变异[51]、部分变异[54],以及高阶变异[52~55]等,并指明了高效的变异体(集合),为使用部分变异算子达到有效的变异测试奠定了基础. 这些方法主要面向 Java 程序或 C/C ++ 程序. 对于其它类型的并发程序,仍缺乏相关的研究. 可以利用已有的串行程序选择变异测试研究成果[76~78],分析并发变异算子之间的包含或制约关系,给出选择变异测试的相关结论. 此外,将并发变异算子与普通变异算子结合,研究高阶变异测试,以模拟与并发和计算相关的缺陷,以及将更多搜索方法应用到并发程序变异测试中,也是提高变异测试效率的可行途径.

(2)按照强变异测试准则,一方面,对并发程序而言,判定变异体是否被杀死需要考虑 2 个输出集合是否相同,这也是并发程序变异测试复杂度高的主要原因. 另一方面,弱变异测试仅考虑变异位置状态的改变,不考虑程序的输出. 应用弱变异测试到并发程序中,是降低测试代价的途径之一. 我们曾研究了消息传递并发程序的弱变异测试[61],基于此,能够利用变异分支占优度约减变异体. 沿着该思路,可以展开其它类型并发程序的弱变异测试研究,分析变异分支的相关性[79],提出弱变异测试的低代价约减策略.

(3)为了克服不确定性对并发程序变异测试的影响,可以采用多次运行或确定执行的方式,这增加了变异测试的代价. 将通信顺序作为测试数据的一部分是另一种解决途径[80],但该方法增大了测试数据生成的搜索空间. Zhang 等提出了预测变异测试,基于构建的分类模型,在不执行变异体的前提下,预测变异体能够被杀死[81]. 等价变异体识别[82]已得到广泛关注,也出现了各种检测等价变异体的策略[83~86]. 然而,已有方法主要适用于串行程序. 如何降低并发程序变异测试的执行代价和高效判定等价变异体,是亟需解决的问题. 另外,将并发程序变异测试问题建模为优化问题,应用有针对性的启发式方法求解,能够进一步改善并发程序的变异测试效率.

(4)并行技术是优化变异测试过程,改善测试效率的重要手段. 虽然有变异测试工具提供了并行化机制,以并行产生同阶变异体[53],但没有进一步讨论并行系统的负载均衡. 针对串行程序,Sun 等将具有相同轨迹的变异体,融合到一个具有并发机制的程序中,避免相同轨迹的重复执行[87]. Cañizares 等也利用高性能集群,降低变异测试的执行时间[43]. 关于并发程序变异测试的负载分配,需要考虑计算节点的并行或线程数、计算节点之间的通信开销. 结合已有的变异测试优化技术,提出合理的负载均衡策略,以降低并发程序变异测试的执行代价,是值得研究的问题.

5　结论

本文介绍了与并发程序和变异测试相关的背景知

识.从变异体产生、变异测试准则、变异测试数据生成3个方面,综述了并发程序变异测试的研究进展,包括:反映并发缺陷的变异算子设计、用于提高变异体生成效率的选择变异,以及高阶变异和局部变异等技术;介绍了变异测试准则的研究现状;评述了变异测试生成、流程交互顺序选择、变异测试执行优化等技术,以及并发变异测试工具;指出了需要进一步研究的问题.

本文重点从并发执行的角度阐述了并发程序变异测试的研究进展,并讨论了可能的研究问题和研究思路.实际上,应用传统的变异算子到并发程序时,同样需要考虑程序不确定执行带来的影响.此外,充分的并发语义变异研究,以及相关的变异实施工具,也是未来研究的重点.

参考文献

[1] Parihar M, Bharti A. Role of software testing life cycle (STLC) in software development life cycle (SDLC) [J]. International Journal of Research, 2019, 6(8): 649 – 661.

[2] 苏小红,禹振,王甜甜,马培军.并发缺陷暴露、检测与规避研究综述[J].计算机学报,2015(11):2215 – 2233.
Su X, Yu Z, Wang T, Ma P. A survey on exposing, detecting and voiding concurrency bugs [J]. Chinese Journal of Computers, 2015, 38(11): 2215 – 2233. (in Chinese)

[3] Bianchi F A, Margara A, Pezzè M. A survey of recent trends in testing concurrent software systems [J]. IEEE Transactions on Software Engineering, 2018, 44(8): 747 – 783.

[4] Christakis M, Sagonas K. Detection of asynchronous message passing errors using static analysis, practical aspects of declarative languages [A]. Proceedings of the International Symposium on Practical Aspects of Declarative Languages [C]. Austin: Springer, 2011. 5 – 18.

[5] Koushik S. Effective random testing of concurrent programs [A]. Proceedings of the 22nd International Conference on Automated Software Engineering [C]. Atlanta: ACM, 2007. 323 – 333.

[6] Godefroid P. Model checking for programming languages using verisoft [A]. Proceedings of the 24th Symposium on Principles of Programming Languages [C]. Paris: ACM, 1997. 174 – 186.

[7] Chen J. Guided Testing of Concurrent Programs Using Value Schedules [D]. Waterloo: University of Waterloo, 2009.

[8] Lipton R. Fault Diagnosis of Computer Programs [R]. Pittsburgh: Carnegie Mellon University, 1971.

[9] DeMillo R A, Lipton R J, Sayward F G. Hints on test data selection: Help for the practicing programmer [J]. Computer, 1978, 11(4): 34 – 41.

[10] Offutt J. A mutation carol: Past, present and future [J]. Information and Software Technology, 2011, 53(10): 1098 – 1107.

[11] Arora V, Bhatia R, Singh M. A systematic review of approaches for testing concurrent programs [J]. Concurrency and Computation: Practice and Experience, 2016, 28(5): 1572 – 1611.

[12] Ma L, Zhang F, Sun J, et al. Deepmutation: mutation testing of deep learning systems [A]. Proceedings of the 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE) [C]. Memphis: IEEE, 2018. 100 – 111.

[13] Habibi E, Mirian-Hosseinabadi S H. Event-driven web application testing based on model-based mutation testing [J]. Information and Software Technology, 2015, 67: 159 – 179.

[14] Kim Y, Hong S. DEMINER: Test generation for high test coverage through mutant exploration [EB/OL]. https://doi.org/10.1002/stvr.1715.2019.10.28.

[15] Carver R, Lei Y. Stateless techniques for generating global and local test oracles for message-passing concurrent programs [J]. Journal of Systems and Software, 2018, 136: 237 – 265.

[16] Yu T, Wen W, Han X, Hayes J. ConPredictor: Concurrency defect prediction in real-world applications [J]. IEEE Transactions on Software Engineering, 2019, 45(5): 558 – 575.

[17] Jia Y, Harman M. An analysis and survey of the development of mutation testing [J]. IEEE Transactions on Software Engineering, 2011, 37(5): 649 – 678.

[18] 陈翔,顾庆.变异测试:原理,优化和应用[J].计算机科学与探索,2012,6(12):1057 – 1075.
Chen X, Gu Q. Mutation testing: Principal, optimization and application [J]. Journal of Frontiers of Computer Science and Technology, 2012, 6(12): 1057 – 1075. (in Chinese)

[19] Madeyski L, Orzeszyna W, Torkar R, Józala M. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation [J]. IEEE Transactions on Software Engineering, 2013, 40(1): 23 – 42.

[20] Souza F C M, Papadakis M, Durelli V H S, Delamaro M E. Test data generation techniques for mutation testing: A systematic mapping [A]. Proceedings of the XVII Ibero-American Conference on Software Engineering (CIBSE2014) [C]. Pucón: University of La Frontera, 2014. 419 – 432.

[21] Silva R A, Souza S R S, Souza P S L. A systematic review on search based mutation testing [J]. Information and Software Technology, 2017, 81: 19 – 35.

[22] Belli F, Budnik C J, Hollmann A, Tuglular T, Wong W. Model-based mutation testing-approach and case studies [J]. Science of Computer Programming, 2016, 12: 25

-48.

[23] Nguyen Q V, Madeyski L. Problems of mutation testing and higher order mutation testing[A]. Proceedings of the 2nd International Conference on Computer Science, Applied Mathematics and Application[C]. Budapest:Springer,2014. 157-172.

[24] Melo S M,Carver J C,Souza P S L,Souza S R S. Empirical research on concurrent software testing:A systematic mapping study[J]. Information and Software Technology,2019,105:226-251.

[25] Pizzoleto A V,Ferrari F C,Offutt J,Fernandes L,Ribeiro M. A systematic literature review of techniques and metrics to reduce the cost of mutation testing[J]. Journal of Systems and Software,2019,157:110388.

[26] Kintis M,Papadakis M,Papadopoulos A,et al. How effective are mutation testing tools? An empirical analysis of Java mutation testing tools with manual analysis and real faults[J]. Empirical Software Engineering,2018,23(4):2426-2463.

[27] Carver R. Mutation-based testing of concurrent programs[A]. Proceedings of the IEEE International Test Conference[C]. Baltimore:IEEE,1993. 845-853.

[28] Jia Y, Harman M. Higher order mutation testing[J]. Information and Software Technology,2009,51(10):1379-1393.

[29] King K N,Offutt A J. A fortran language system for mutation based software testing[J]. Software:Practice and Experience,1991,21(7):685-718.

[30] 吴迪,徐宝文. Ada 语言的发展[J]. 计算机科学,2014,41(1):1-15.
Wu D,Xu B. Evolution of Ada programming language[J]. Computer Science,2014,41(1):1-15. (in Chinese)

[31] Offutt A J, Voas J, Payne J. Mutation Operators for Ada[R]. Virginia :George Mason University,1996.

[32] Ghosh S. Towards measurement of testability of concurrent object-oriented programs using fault insertion:a preliminary investigation[A]. Proceedings of the 2nd IEEE International Workshop on Source Code Analysis and Manipulation[C]. Montreal:IEEE,2002. 17-25.

[33] Delamaro M,Pezzè M,Vincenzi A M R,Maldonado J C. Mutant operators for testing concurrent Java programs[A]. Proceedings of the Brazilian Symposium on Software Engineering[C]. Rio de Janeiro:Brazilian Computer Society,2001. 272-285.

[34] Bradbury J S,Cordy J R,Dingel J. Mutation operators for concurrent Java(J2SE 5.0)[A]. Proceedings of the 2nd Workshop on Mutation Analysis[C]. Raleigh: IEEE, 2006. 83-92.

[35] Wu X,Zheng W,Shi Z,Wang Z,Cao L,Mu D. Concurrency bug-oriented mutation operators design for Java[A]. Proceedings of the 2018 IEEE International Conference on Progress in Informatics and Computing(PIC)[C]. Suzhou:IEEE,2018. 364-369.

[36] Benini L, Bertozzi D, Bogliolo A, Menichelli F, Olivieri M. Mparm: Exploring the multi-processor soc design space with systemc[J]. Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology, 2005,41(2):169-182.

[37] Sen A. Mutation operators for concurrent SystemC designs[A]. Proceedings of 2009 10th International IEEE Workshop on Microprocessor Test and Verification[C]. Austin:IEEE,2009. 27-31.

[38] Nilsson R, Offutt J, Andler S F. Mutation-based testing criteria for timeliness[A]. Proceedings of the 28th Annual International Computer Software and Applications Conference[C]. Hong Kong:IEEE,2004. 306-311

[39] Agha G,Houck C,Panwar R. Distributed execution of actor programs[A]. Proceedings of the International Workshop on Languages and Compilers for Parallel Computing[C]. Santa Clara:Springer,1991. 1-17.

[40] Jagannath V,Gligoric M,Lauterburg S,Marinov D,Agha G. Mutation operators for actor systems[A]. Proceedings of the 2010 3rd International Conference on Software Testing, Verification, and Validation Workshops[C]. Paris: IEEE,2010. 157-162

[41] Snir M,Otto S,Steven H,Walker D,Dongarra J. MPI-the Complete Reference: The MPI Core[M]. USA: MIT Press,1998.

[42] Silva R A,Souza S R S,Souza P S L. Mutation operators for concurrent programs in MPI[A]. Proceedings of the 2012 13th Latin American Test Workshop(LATW)[C]. Quito:IEEE,2012. 1-6.

[43] Cañizares P C,Núñez A,Lara J. OUTRIDER:Optimizing the mutation testing process in distributed Environments[J]. Procedia Computer Science,2017,108:505-514.

[44] Estero-Botaro A,Palomo-Lozano F,Medina-Bulo I. Mutation operators for WS-BPEL 2.0[A]. Proceedings of the 21th International Conference on Software & Systems Engineering and their Applications[C]. Paris: Genie Logiciel,2008. 1-7.

[45] Clark J A,Dan H,Hierons R M. Semantic mutation testing[J]. Science of Computer Programming, 2013, 78(4):345-363.

[46] Cao L,Zheng W,Hu D,Bai H. Concurrent program semantic mutation testing based on abstract memory model[A]. Proceedings of the 2015 IEEE International Conference on Information and Automation[C]. Lijiang:IEEE, 2015. 1200-1205.

[47] 钱葭南,王雅文,宫云战,等. 一种变异测试中冗余变异体的寻找方法[J]. 电子学报,2017,45(8):1970 – 1975.
Qian Gen-nan, Wang Ya-wen, Gong Yun-zhan. A method four finding redundant mutants in mutation testing[J]. Acta Electronica Sinic, 2017, 45(8): 1970 – 1975. (in Chinese)

[48] Papadakis M, Kintis M, Zhang J, Jia Y, LeTraon Y, Harman M. Mutation testing advances: An analysis and survey [J]. Advances in Computers. Netherlands: Elsevier, 2019, 112: 275 – 378.

[49] Gopinath R, Alipour A, Ahmed I, Jensen C, Groce A. How hard does mutation analysis have to be, anyway? [A]. Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering (ISSRE) [C]. Gaithersbury: IEEE, 2015. 216 – 227.

[50] Namin A S, Andrews J H, Murdoch D J. Sufficient mutation operators for measuring test effectiveness[A]. Proceedings of the 30th International Conference on Software engineering[C]. Leipzig: IEEE, 2008. 351 – 360.

[51] Gligoric M, Zhang L, Pereira C, Pokam G. Selective mutation testing for concurrent code[A]. Proceedings of the 2013 International Symposium on Software Testing and Analysis[C]. Lugano: ACM, 2013. 224 – 234.

[52] Wu L, Kaiser G. Constructing subtle concurrency bugs using synchronization-centric second-order mutation operators[R]. New York: Columbia University Computer Science Technical Reports, 2011.

[53] Kusano M, Wang C. CCmutator: A mutation generator for concurrency constructs in multithreaded C/C ++ applications[A]. Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering [C]. Palo Alto: IEEE, 2013. 722 – 725.

[54] Madiraju P, Namin A S. Paraμ—A partial and higher-order mutation tool with concurrency operators[A]. Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops[C]. Berlin: IEEE, 2011. 351 – 356.

[55] 孙昌爱,王真,潘琳. 面向 WS-BPEL 程序的变异测试优化技术[J]. 计算机研究与发展,2019,56(4):895 – 905.
Sun Changai, Wang Zhen, Pan Lin. Optimized mutation testing techniques for WS-BPEL programs[J]. Journal of Computer Reasearch and Developlent, 2019, 56(4): 895 – 905. (in Chinese)

[56] DeMilli R A, Offutt A J. Constraint-based automatic test data generation[J]. IEEE Transactions on Software Engineering, 1991, 17(9): 900 – 910.

[57] Howden W E. Weak mutation testing and completeness of test sets[J]. IEEE Transactions on Software Engineering, 1982, 8(4): 371 – 379.

[58] Marick B. The weak mutation hypothesis[A]. Proceedings of the Symposium on Testing, Analysis, and Verification[C]. Victoria: ACM, 1991. 190 – 199.

[59] Offutt A J, Lee S D. An empirical evaluation of weak mutation[J]. IEEE Transactions on Software Engineering, 1994, 20(5): 337 – 344.

[60] Papadakis M, Malevris N. Automatically performing weak mutation with the aid of symbolic execution, concolic testing and search-based testing[J]. Software Quality Journal, 2011, 19(4): 691 – 723.

[61] 巩敦卫,陈永伟,田甜. 消息传递并发程序的弱变异测试及其转化[J]. 软件学报,2016,27(8):2008 – 2024.
Gong D, Chen Y, Tian T. Weak mutation testing and its transformation for message passing parallel programs[J]. Journal of Software, 2016, 27(8): 2008 – 2024. (in Chinese)

[62] Singh M, Srivastava V M. Extended firm mutation testing: A cost reduction technique for mutation testing[A]. Proceedings of the 2017 Fourth International Conference on Image Information Processing (ICIIP) [C]. Waknaghat: IEEE, 2017. 1 – 6.

[63] Woodward M R, Halewood K. From weak to strong, dead or alive[A]. Proceedings of the Second Workshop on Software Testing, Verification and Analysis[C]. Banff: IEEE, 1988. 152 – 158.

[64] Palomo-Lozano F, Estero-Botaro A, Medina-Bulo I, Núñez M. Test suite minimization for mutation testing of WS-BPEL compositions[A]. Proceedings of the Genetic and Evolutionary Computation Conference[C]. Kyoto: ACM, 2018. 1427 – 1434.

[65] Estero-Botaro A, Palomo-Lozano F, Medina-Bulo I, Domínguez-Jiménez J J, García-Domínguez A. Quality metrics for mutation testing with applications to WS-BPEL compositions[J]. Software Testing, Verification and Reliability, 2015, 25(5 – 7): 536 – 571.

[66] Gligoric M, Jagannath V, Marinov D. MuTMuT: Efficient exploration for mutation testing of multithreaded code [A]. Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation [C]. Paris: IEEE, 2010. 55 – 64.

[67] Sen A, Abadir M S. Coverage metrics for verification of concurrent SystemC designs using mutation testing[A]. Proceedings of the 2010 IEEE International High Level Design Validation and Test Workshop (HLDVT) [C]. Anaheim: IEEE, 2010. 75 – 81.

[68] Harman M, Jones B F. Search-based software engineering [J]. Information and software Technology, 2001, 43(14): 833 – 839.

[69] Ozkaya I. The golden age of software engineering[J]. IEEE Software, 2019(1): 4 – 10.

[70] Khari M, Kumar P. An extensive evaluation of search-based software testing: a review[J]. Soft Computing, 2019,23(6):1933 – 1946.
[71] Rodrigues D S, Delamaro M E, Corrêa C G, Nunes F L S. Using genetic algorithms in test data generation: A critical systematic mapping[J]. ACM Computing Surveys (CSUR), 2018,51(2):41:1 – 41:23.
[72] Silva R A. Search Based Software Testing for the Generation of Synchronization Sequences for Mutation Testing of Concurrent Programs[D]. São Paulo: Universidade de São Paulo, 2018.
[73] Nilsson R, Offutt J, Mellin J. Test case generation for mutation-based testing of timeliness[J]. Electronic Notes in Theoretical Computer Science, 2006, 164(4):97 – 114.
[74] Ghiduk A S, El-Zoghdy S F. CHOMK: concurrent higher-order mutants killing using genetic algorithm[J]. Arabian Journal for Science and Engineering, 2018, 43(12):7907 – 7922.
[75] Takagi T, Arao T. Overview of a place/transition net-based mutation testing framework to obtain test cases effective for concurrent software[A]. Proceedings of the 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)[C]. Takamatsu: IEEE, 2015. 1 – 3.
[76] Delgado-Pérez P, Segura S, Medina-Bulo I. Assessment of C++ object-oriented mutation operators: A selective mutation approach[EB/OL]. https://doi.org/10.1002/stvr.1630.2017.03.20.
[77] Kurtz B, Ammann P, Offutt J, Delamaro M E, Kurtz M, GökCçe N. Analyzing the validity of selective mutation with dominator mutants[A]. Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering[C]. Seattle: ACM, 2016. 571 – 582.
[78] Zhang J. Scalability studies on selective mutation testing[A]. Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering[C]. Florence: IEEE, 2015. 851 – 854.
[79] Gong D, Yao X, Xia M. Automatic determination of branch correlations in software testing[A]. Proceedings of the 2009 WRI World Congress on Software Engineering[C]. Xiamen: IEEE, 2009. 211 – 215.
[80] Tian T, Gong D, Kuo F C, Liu H. Genetic algorithm based test data generation for MPI parallel programs with blocking communication[J]. Journal of Systems and Software, 2019, 155:130 – 144.
[81] Zhang J, Zhang L, Harman M, Hao D, Jia Y, Zhang L. Predictive mutation testing[J]. IEEE Transactions on Software Engineering, 2019, 45(9):898 – 918.
[82] Madeyski L, Orzeszyna W, Torkar R, Jozala M. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation[J]. IEEE Transactions on Software Engineering, 2013, 40(1):23 – 42.
[83] Kintis M, Malevris N. Using data flow patterns for equivalent mutant detection[A]. Proceedings of the 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops[C]. Cleveland: IEEE, 2014. 196 – 205.
[84] Kintis M. Effective Methods to Tackle the Equivalent Mutant Problem When Testing Software with Mutation[D]. Athens: Athens University of Economics and Business, 2016.
[85] Holling D, Banescu S, Probst M, Petrovska A, Pretschner A. Nequivack: Assessing mutation score confidence[A]. Proceedings of the 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)[C]. Chicago: IEEE, 2016. 152 – 161.
[86] Papadakis M, Malevris N. Mutation based test case generation via a path selection strategy[J]. Information and Software Technology, 2012, 54(9):915 – 932.
[87] Sun C, Jia J, Liu H, Zhang X. A lightweight program dependence based approach to concurrent mutation analysis[A]. Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)[C]. Tokyo: IEEE, 2018. 116 – 125.

作者简介

田　甜　女,1987 年 1 月出生于山东德州,山东建筑大学副教授,主要研究方向为程序分析与测试.
E-mail:tian_tiantian@126.com

巩敦卫(通信作者)　男,1970 年 3 月出生于江苏铜山,中国矿业大学教授,博士生导师,主要研究方向:基于搜索的软件工程、智能优化与控制.
E-mail:dwgong@vip.163.com