软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(1):130−149 [doi: 10.13328/j.cnki.jos.006425]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

蜕变测试研究进展及其在并行程序测试中的研究展望*

田 甜[1], 杨秀婷[1], 王安轼[1], 于 旭[2], 巩敦卫[3]

[1](山东建筑大学 计算机科学与技术学院, 山东 济南 250101)

[2](青岛科技大学 信息科学技术学院, 山东 青岛 266061)

[3](中国矿业大学 信息与控制工程学院, 江苏 徐州 221116)

通信作者: 巩敦卫, E-mail: dwgong@vip.163.com

摘 要: 在软件测试过程中, 待测程序的预期输出是判断软件是否存在缺陷的重要因素. 蜕变测试技术是利用被测软件的属性来检查程序输出, 从而有效地解决程序预期输出难以构造的问题. 近年来, 蜕变测试在软件测试领域取得了蓬勃的发展, 许多研究人员将蜕变测试技术进行优化, 将其运用到各个领域, 有效提高了软件质量. 从原理、过程及其优化, 应用领域 3 个方面, 总结蜕变测试的研究工作, 着重分析了近 5 年的研究进展, 进一步展望了蜕变测试用于并行程序时, 可能的研究主题. 首先, 介绍蜕变测试的基本概念和蜕变测试过程; 接着, 从蜕变关系、测试用例、测试执行过程以及蜕变测试工具 4 个角度, 总结蜕变测试优化技术; 然后, 汇总了蜕变测试的应用领域; 最后, 基于已有研究成果, 讨论蜕变测试在并行程序测试领域面临的问题, 为蜕变技术在并行程序测试领域的研究提供可能的思路.

关键词: 蜕变测试; 蜕变关系; 原始测试用例; 蜕变测试优化; 并行程序; 蜕变测试应用

中图法分类号: TP311

中文引用格式: 田甜, 杨秀婷, 王安轼, 于旭, 巩敦卫. 蜕变测试研究进展及其在并行程序测试中的研究展望. 软件学报, 2023, 34(1): 130–149. http://www.jos.org.cn/1000-9825/6425.htm

英文引用格式: Tian T, Yang XT, Wang AS, Yu X, Gong DW. Research Progress of Metamorphic Testing and Its Research Prospects in Parallel Program Testing. Ruan Jian Xue Bao/Journal of Software, 2023, 34(1): 130–149 (in Chinese). http://www.jos.org.cn/1000-9825/6425.htm

Research Progress of Metamorphic Testing and Its Research Prospects in Parallel Program Testing

TIAN Tian[1], YANG Xiu-Ting[1], WANG An-Shi[1], YU Xu[2], GONG Dun-Wei[3]

[1](School of Computer Science and Technology, Shandong Jianzhu University, Jinan 250101, China)

[2](College of Information Science and Technology, Qingdao University of Science and Technology, Qingdao 266061, China)

[3](School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China)

Abstract: In the process of software testing, the expected output of a program under test is an important factor in judging whether the program is defective or not. Metamorphic testing technique uses the properties of the program under test to check the output of the program, so as to effectively solve the problem of being difficult to construct the expected output of the program. In recent years, metamorphic testing has blossomed in the field of software testing. Many researchers have optimized techniques related to metamorphic testing and applied them to various fields to effectively improve software quality. This study summarizes and analyzes the research work of metamorphic testing from the following three aspects: theoretical knowledge, improvement strategies and application areas, and focuses on the research results of the past five years. Meanwhile, the potential research is discussed when metamorphic testing is applied for parallel programs. First, the basic concepts of metamorphic testing and the metamorphic testing process are provided. Next, according to its steps, the optimization techniques for metamorphic testing are summarized from the four perspectives: metamorphic relationships, test

* 基金项目: 山东省自然科学基金 (ZR2020MF084); 国家自然科学基金 (61773384)
收稿时间: 2021-03-26; 修改时间: 2021-06-22; 采用时间: 2021-08-04; jos 在线出版时间: 2021-10-20
CNKI 网络首发时间: 2022-11-15

case generation, test execution, and metamorphic testing tools. Then, the application fields of metamorphic testing are listed. Finally, based on the existing research results, the problems faced by metamorphic testing are discussed in parallel program testing, and the possible solutions are provided for further research.

Key words: metamorphic testing (MT); metamorphic relationship; original test cases; optimization of metamorphic testing; parallel program; metamorphic testing application

软件测试是软件质量保证的重要方法. 传统的软件测试技术将测试用例的执行结果与预期结果进行对比, 从而发现软件中的错误. 然而, 对于复杂的数值计算程序, 预测正确的程序输出比较困难, 使得传统测试策略不总是可行的, 该问题被称为 Oracle 问题[1], 也就是说, 测试人员无法预估测试用例的期望输出. 为了有效地解决 Oracle 问题, Chen 等人[1]引入了蜕变测试 (metamorphic testing, MT) 的概念. 蜕变测试技术通过验证程序执行结果是否满足某一个特定的“关系”来判断其是否正确, 该技术能够有效解决预期结果难以构造的问题[1,2].

蜕变测试概念的提出得到了国内外学者的广泛关注. 研究人员对其展开了深入的探索, 并将其广泛地应用于机器学习[3]、密码学[4]、编译器[5]、软件系统[6]、自然语言处理[7]、数据集质量评估[8]等领域. 一方面, 蜕变测试技术可以检测程序故障, 例如, MT 在西门子套件[9]的 7 个程序中检测到了之前未知的 3 个错误; 另一方面, 可以评估软件质量特征[10], 例如, 软件的正确性、容量、性能、有效性等. 蜕变测试已经发展成为软件验证、检测和质量评估的统一框架[10].

董国伟等人[11]从蜕变测试的 7 个方面综述了蜕变测试早期的研究成果. Segura 等人[12]首先就蜕变关系提出 4 个问题, 然后从“不同维度分析蜕变测试相关工作发表成果数据”“蜕变测试主要进展”“应用领域”和“论文实验分析”这 4 个方向作为解决所提问题的切入点, 全面综述了 1998－2015 年间发表的 119 篇论文的概况. Chen 等人[13]从不同的角度总结分析了相关研究, 并进一步解释了蜕变测试的概念及其挑战等. 近几年, 以机器学习与人工智能技术为代表的新兴技术, 使蜕变测试本身得到了很好的发展的同时, 也让蜕变技术应用于更多新兴领域的软件测试. 与已有研究不同, 本文在已有蜕变测试综述研究的基础上, 从蜕变测试原理、蜕变测试过程及其优化等方面, 对蜕变测试研究进展进行深入的总结, 重点阐述了近 5 年国内外的研究工作较早期成果的改进. 进一步, 结合最新提出的相关概念和并行程序特征, 探讨并行程序蜕变测试可能的研究主题.

本文第 1 节介绍了文献情况; 第 2 节阐述蜕变测试原理和蜕变测试过程; 第 3 节汇总了蜕变测试优化技术; 第 4 节总结分析了蜕变测试的应用; 第 5 节讨论了将蜕变技术用于并行程序测试时, 可能面临的问题和解决思路; 第 6 节总结全文.

1 文献情况

1.1 搜索策略

本文文献收集时间段分为两部分: (1) 2016 年 1 月 1 日到 2021 年 6 月期间发表的计算机及其相关学科论文; (2) 为了阐述近 5 年较 2016 年前研究成果的进展, 对从蜕变测试概念第一次提出的 1998 年到 2015 年 12 月 30 日期间发表的论文进行了检索. 在学术资源数据库中, 搜索蜕变测试、蜕变关系、蜕变测试工具、蜕变测试应用、oracle、metamorphic testing、metamorphic relations、original test cases、metamorphic testing tools、application of metamorphic testing 等关键字, 检索与蜕变测试有关的论文. 同时, 逐一排查论文的参考文献中是否存在与蜕变测试相关且被遗漏的文献.

将按照上述检索方法获取的论文成果进行筛选. 首先, 将截至 2021 年 6 月的论文成果进行整理, 排除重复和没有明显差异的文献; 然后, 由于现有文献[12]比较全面地汇总了截至 2016 年的蜕变测试应用领域, 本文还排除了 2016 之前, 仅应用蜕变测试理念, 但没有对蜕变测试过程做任何改进的论文. 经过上述筛选, 本文一共获取了有关蜕变测试的论文 100 篇. 为了简化描述, 截至 2015 年 12 月以及在已有综述中汇总的研究成果, 称之为早期研究成果.

1.2 文献分布

在选取的 100 篇论文中, 2016 年以后出版的占 67 篇, 33 篇是 2016 年之前发表的论文. 论文分布如表 1 所示. 表 2 和表 3 汇总了两篇及以上论文的期刊与会议名称, 这些论文的研究方向分布情况如表 4 所示.

表 1 论文分布

地区	论文数量	类型	论文数量
国内	16	期刊	11
		学位	5
国外	84	期刊	34
		会议	50

表 2 期刊名称

名称	论文数量
IEEE Transactions on Software Engineering	6
Journal of Systems and Software	4
IEEE Transactions on Reliability	3
Software Quality Journal	2
Software: Testing, Verification and Reliability	2
Information and Software Technology	3

表 3 会议名称

名称	论文数量
International Workshop on Metamorphic Testing	17
International Computer Software and Applications	3
International Symposium on Software Reliability Engineering	2
International Conference on Software Engineering	2
International Conference on Quality Software	3
International Workshop on Automation of Software Test	2
Artificial Intelligence Testing	2

表 4 论文研究方向分布情况表

时期	研究方向	论文数量
早期	蜕变关系识别与选取	18
	原始测试用例生成	7
	蜕变测试工具开发	5
近期	蜕变关系识别与选取	17
	原始测试用例生成	3
	蜕变测试工具开发	4
	蜕变测试应用拓展	43

2 蜕变测试原理

2.1 蜕变测试概念

在软件测试的过程中, 不能检测出程序错误的测试用例称为“成功的测试用例”[1], 一般被认为没有价值. Chen 等人[1]提出该类测试用例体现程序的部分特征, 可以融入到程序的检测中. 1998 年, Chen 等人[1]正式提出了蜕变测试的概念.

蜕变测试的基本原理是: 利用程序输入-输出关系, 将已有的测试用例转化为新的测试用例用以测试程序, 其中, 程序输入-输出关系即为蜕变关系, 生成的新的测试用例即为衍生测试用例. 用原始和衍生测试用例分别执行程序, 检验相应的输出是否满足构造的蜕变关系. 如果不满足, 说明程序存在故障. 蜕变测试技术的提出, 充分利用了成功的测试用例, 有效解决了测试人员难以构造测试用例输出问题.

定义 1. 蜕变关系 (metamorphic relation, MR)[2]. 假设 S 是实现函数 f 的程序, $x_1, x_2, \ldots, x_n$ 是 f 的 n 组输入变量, $f(x_1), f(x_2), \ldots, f(x_n)$ 是对应的输出结果, 若 $x_1, x_2, \ldots, x_n$ 满足关系 r 时, $f(x_1), f(x_2), \ldots, f(x_n)$ 满足关系 r_f 即:

$$r(x_1, x_2, \ldots, x_n) \Rightarrow r_f(f(x_1), f(x_2), \ldots, f(x_n)) \tag{1}$$

称 (r, r_f) 是 S 的蜕变关系. 显然, 如果 S 是正确的, 那么一定满足:

$$r(I_1, I_2, \ldots, I_n) \Rightarrow r_f(S(I_1), S(I_2), \ldots, S(I_n)) \tag{2}$$

其中, $I_1, I_2, \ldots, I_n$ 是程序 S 的对应函数的输入, $S(I_1), S(I_2), \ldots, S(I_n)$ 是对应的输出.

在已有研究的基础上, 惠战伟等人[14]提出一种蜕变关系描述方法, 给出蜕变关系的如下延伸定义:

$$(P(x_i) = [P](x_i))/(r(x_1, x_2, \ldots, x_n) \Rightarrow r_f([P](x_1), [P](x_2), \ldots, [P](x_n))) \tag{3}$$

其中, $x_1, x_2, \ldots, x_n$ 为 n 组输入变量; $[P](x_1), [P](x_2), \ldots, [P](x_n)$ 为函数 $[P]$ 相应输出; $r(x_1, x_2, \ldots, x_n)$ 为输入之间的关系; $r_f([P](x_1), [P](x_2), \ldots, [P](x_n))$ 为输出之间的关系. 根据分子和分母的不同取值, 对蜕变关系的有效性进行分析. 假设 $P(x_i)=[P](x_i)$ 成立, 即分子取值为 1, 则蜕变关系定义式存在 4 种可能, 如表 5 所示. 其中, 0 代表某类关系不满足, 1 代表满足某类关系, 例如: 第 3 行代表输入关系不满足, 输出关系满足, 程序满足蜕变关系, 无论任何关系的测试输入, 蜕变关系均满足, 因此该类蜕变关系是没有作用的. 在公式 (3) 所有可能取值中, 第 1 和 2 类蜕变关系可以用于验证程序正确性和检测程序错误; 第 3 和 4 类蜕变关系则为无效的蜕变关系. 借助此描述方法, 能够更好地理解蜕变关系与软件故障的关系.

表 5 蜕变关系定义式取值分析

取值	输入关系	输出关系	MR	含义
1/(1⇒1)	√	√	√	程序没有错误
1/(1⇒0)	√	×	×	程序存在错误
1/(0⇒1)	×	√	√	MR无意义
1/(0⇒0)	×	×	×	MR无意义

由于缺乏一个标准的机制来描述蜕变关系, 导致不同程序蜕变关系的描述方法也不同, 不利于测试人员之间的沟通. 基于此, Segura 等人[15]从软件度量领域使用的目标、问题和度量模式中得到启发, 提出了一种基于模板的蜕变关系描述方法, 本质上, 是一种包含应用领域、上下文定义、约束限制、蜕变关系名称、蜕变关系输入与输出内容的固定描述模板. 根据该模板描述不同项目的蜕变关系, 能够简化测试人员的交流.

定义 2. 原始/衍生测试用例[2]. 使用蜕变关系 (r, r_f) 测试程序 S 时, 已知的测试用例, 即成功的测试用例, 是原始测试用例; 由原始测试用例根据关系 r 转化的新测试用例, 称为基于蜕变关系 (r, r_f) 的衍生测试用例.

2.2 蜕变测试过程

基于蜕变测试原理可知, 蜕变测试过程主要分为 4 个部分[2]: (1) 筛选原始测试用例; (2) 构造一组蜕变关系; (3) 基于蜕变关系将原测试用例转化成衍生测试用例; (4) 用原始与衍生测试用例两次执行程序, 判断程序的正确性.

例如, 程序 S 的功能是计算函数 $g(x)$=arctan(x) 的值. 已知程序输入是 x_1=5, 因为该数值型函数预估其输出难度比较大, 所以将蜕变测试用于检测其输出结果的正确性. 根据该函数是奇函数的性质, 设计如下蜕变关系:

$$\arctan(x_1) = -\arctan(x_2).$$

然后, 生成衍生测试用例 $x_2 = -5$, 将其分别执行程序, 输出结果若满足 arctan(5) = −arctan(−5), 则代表没有发现程序 S 的故障.

此外, 使用谷歌翻译、百度翻译和有道翻译等不同的翻译软件, 可能会产生不同的结果, 可以采用蜕变测试技术来衡量翻译软件的质量. 设计蜕变关系为: 设 L_1 与 L_2 为两种不同的语言, X_1 与 X_2 是用两种语言写的信息, A 为情感分析工具. 如果 X_2 是 X_1 的翻译, 那么 A 对于 X_1 与 X_2 的情感分析应该是相似的[7].

可以将蜕变测试总结为“两元素、一过程”. 其中, 两元素是蜕变关系和测试用例, 连接这两元素的是蜕变测试执行过程. 目前的研究一般从这 3 方面进行优化, 第 3 节对蜕变测试改进策略进行总结分析.

3 蜕变测试过程优化

3.1 蜕变关系识别与选取

蜕变关系领域主要包括: 蜕变关系的识别与选择[11]. 不同的蜕变关系有不同的检错能力, 选择检错能力较强的蜕变关系在蜕变测试过程中至关重要.

3.1.1 蜕变关系识别

早期蜕变关系的识别主要来源于测试人员对程序的理解, 手动生成蜕变关系. 随着蜕变测试的发展, 逐渐出现了一些新的有效方法, 例如, 复合蜕变关系的生成、基于搜索的蜕变关系推理、机器学习方法的蜕变关系识别等[12]. 表 6 汇总了早期蜕变关系的生成策略[16−24]. 其中文献 [24] 主要集成了已有的早期开发的工具, 故将其归到早期研究成果. 文献 [25,26] 属于同一方法, 文献 [25] 是对文献 [26] 的改进, 因此将文献 [25,26] 作为近期研究成果.

然而, 蜕变关系识别方法依然存在一些缺陷, 主要表现在: 研究人员需要从头开始针对每个被测程序识别 MR. 程序不同, 识别方法则不同. 系统的识别方法的缺乏, 降低了蜕变测试的效率. 因此, 多个蜕变关系的自动识别对于蜕变测试效率的提高有着重要的作用. 近 5 年, 学者们结合表 6 中关于早期蜕变关系识别的研究成果, 提出一些新的策略, 改进了蜕变关系生成方法, 主要分为以下方面.

(1) 蜕变关系模式

Zhou 等人[27]利用多个抽象形式的 MR, 形成多个具体的 MR. 并将原始用例输出与衍生用例输出之间的子集

关系作为一种抽象关系. 通过这种方式生成的蜕变关系, 利用了程序逻辑的一致性, 缓解了搜索引擎数据质量无法测量的缺陷. Segura 等人[28]引入了“蜕变关系输出模式 (metamorphic relation output patterns, MROP)”, 根据程序输出之间相等、并、交和补的集合操作, 确定了如下 MROP: 等价性、相等性、子集、不相交和完全. 实验研究发现, 该模式有助于生成具体的 MR. 例如, 使用百度学术搜索论文时, 无论采用什么样的排序标准, 返回的论文集合, 应该是相等的. 但是, 该模式只研究了输出之间的关系, 且输出的表现形式必须是集合, 只能在特定类型的程序中使用, 例如, 搜索函数, 局限了适用范围. 为了改进上述缺陷, Zhou 等人[29]定义了一种蜕变关系模式 (metamorphic relation patterns, MRP), 代表抽象的一组蜕变关系, 并提出了“对称”蜕变关系模式和“改变方向”蜕变关系输入模式, 改进了 MROP 的缺陷, 可以用来推导出多个不同领域的蜕变关系, 对于跟搜索有关程序的蜕变测试, 既简单又高效. 接着, 又提出了“噪声”蜕变关系模式[30], 它是 MRP 下的子模式, 实现了 MRP 的拓展. 他们的工作为广义的“蜕变关系模式”开辟了新的研究方向.

表 6 蜕变关系生成策略

	方法	优点	不足
早期	复合蜕变关系[18,19]	(1) 减少MR和测试用例执行次数 (2) 能够在较高断层探测效能	(1) MR是否满足组合条件 (2) 若进行组合的MR关联性较弱, 错误检测率的改进较小
	机器学习[16,17]	能够预测程序是否满足特定MR	(1) 代表程序特性的特征不充分 (2) 仅预测3种特定的MR (3) 一次实验只能预测一个MR
	搜索方法[20−23]	自动生成MR	适用有限的程序类型
	METRIC工具[24]	通过定义包含所有输入组合的抽象测试用例, 指导测试人员识别蜕变关系	(1) MR识别依赖测试人员的专业知识与经验 (2) 仅适用于输入域可以划分的程序
近期	蜕变关系模式[27−30]	根据抽象模型可以生成许多具体蜕变关系	仅适用于面向用户的搜索类程序
	机器学习方法[31−33]	(1) 特征更充分 (2) 同时预测多个蜕变关系	仅能够预测5种常见的蜕变关系
	复合蜕变关系[34,35]	(1) 无须进行静态分析 (2) 提供生成高检错率复合蜕变关系的所需条件	待组合蜕变关系需手动生成
	代数规格和错误类型[25,26]	提高了面向对象程序测试的检错率	仅适用于面向对象程序

(2) 机器学习自动预测

随着机器学习和人工智能的发展, 越来越多的研究者将机器学习运用到软件测试的各个方面, 并尝试采用机器学习方法自动预测指定的 MR. 表 6 中文献 [16,17] 首次在函数级程序的蜕变关系预测中使用了机器学习技术, 首先, 提取代表程序的信息、控制流图 (control flow graph, CFG) 中的节点特性, 以及路径执行轨迹, 构成特征数据集; 然后, 将该数据集作为支持向量机 (support vector machine, SVM) 预测模型的输入, 并训练预测模型; 最后, 采用该模型预测未知函数的蜕变关系. 该方法能够预测 3 种特定类型的关系: 置换关系、加性关系和包含关系. 进一步, Kanewal 等人[31]对其进行了改进, 使用图形内核机器学习方法预测蜕变关系, 该方法计算图形内核值, 建立预测模型预测未知函数可能满足的 MR.

上述研究建立的预测模型只能预测一个蜕变关系. 虽然缓解了手动建立蜕变关系的难度, 但是大多数函数存在多个 MR, 因此, MR 的构造效率仍然不高. 鉴于此, Zhang 等人[32]从控制流图中提取代表函数的特征, 用多标签代替单标签, 使用 RBF (radial basis function) 神经网络自动预测函数可能的多个蜕变关系; 另外, 通过将 RBF 神经网络预测模型代替支持向量机预测模型, 改进了预测精度.

穆翔宇等人[33]将机器学习算法运用到多项式蜕变关系的自动生成. 首先, 将程序的输入与输出变量定义为线性关系, 得到关于蜕变关系的损失函数; 然后, 采用 Adam 算法进行优化, 减少算法学习率取值对梯度下降算法的影响; 最后, 采用梯度下降算法求解蜕变关系损失函数的最小值, 进而确定损失函数的系数, 实现蜕变关系的自动

提取.

(3) 复合蜕变关系

早期对于复合蜕变关系的研究, 需要静态分析蜕变关系是否满足组合策略. 为了改进这一缺陷, Xiang 等人[34]提出了一种基于遗传算法的程序复合蜕变关系自动构造方法. 该研究使用一组关系序列来表示一类特定的 MR, 将生成复合 MR 问题转化为寻找合适序列的问题, 并使用遗传算法搜索最佳的关系序列集.

如果 MR_x 可以与 MR_y 组合为复合 MR_{xy}, 则称 MR_x 和 MR_y 为 MR_{xy} 的组件 MR. 研究表明, 复合 MR 可能比组件 MR 故障检测能力要低, 为了研究组件 MR 的特征, 生成故障检测能力高的复合 MR, Qiu 等人[35]探索出, 若 MR_x、MR_y 具备这样的关系: MR_x 的输出映射是内射的, MR_y 的输入映射是从 MR_y 的原输入到 MR_x 原输入的双向映射. 则由 MR_x、MR_y 复合的 MR 会有比较高效的故障检测率, 该工作为生成和选择高故障检测率的蜕变关系提供了新的思路.

近几年, 学者们还针对面向对象软件构造蜕变关系. 封装性、继承性、多态性是面向对象编程的主要特性, 使得绝大数面向过程的蜕变测试方法, 不适用于面向对象软件的测试. 基于此, 张兴隆等人[25]根据面向对象程序的错误点、错误引发的结果和程序的执行结果 3 方面, 将程序的错误进行分类, 基于不同的错误类型生成不同的蜕变关系. 实验证明, 该方法生成的蜕变关系既可以检测程序错误, 也可以实现错误定位. 侯雪梅等人[26]将蜕变关系与 GFT (generating a finite number of test cases) 算法相结合, 考虑类的代数规格, 包括类集合、方法集合和等式公理集合, 构造少量且有效的蜕变关系, 进而提高蜕变测试效率.

本节主要补充了在蜕变关系生成方面的研究进展, 并与早期蜕变关系生成策略进行比较, 如表 6 所示. 主要包括蜕变关系模式、基于机器学习的蜕变关系预测、生成复合蜕变关系和面向对象软件的蜕变关系构造等.

3.1.2 蜕变关系选取

在软件测试过程中, 需要构造大量的 MR 以保证测试有效性. 不同蜕变关系的检测能力存在差异. 如果这些蜕变关系被全部使用, 将产生大量冗余测试用例, 严重影响测试效率. 因此, 有必要选取检错率高的蜕变关系. Segura 等人[12]总结了良好蜕变关系的特性, 如表 7 中早期提出的有效蜕变关系具备的特性所示[36−49].

表 7 有效蜕变关系具备的特性

时期	文献	特性
早期	[46]	包含程序结构
	[39,40]	衍生与原测试用例执行路径差异大
	[47,48]	针对程序局部功能
	[36]	测试用例具有高覆盖率
	[49]	正确、构造衍生测试用例复杂度低、表达式复杂、涉及输出变量多和衍生测试用例唯一
近期	[37,38]	数据集覆盖率高
	[41,42]	测试用例路径对之间差异度和参数差异度大
	[43]	基于蜕变测试用例的变异得分高
	[43,44]	包含程序的多个功能
	[45]	结合上下文特征和执行结果

由于 MR 质量将直接影响故障检测高效性, 因此, 研究人员结合 Segura 等人总结的有效蜕变关系具备的特性, 从故障检测的角度进行了研究.

表 7 中文献 [36] 评估了不同蜕变关系覆盖率的故障检测效果, 结果表明: 高覆盖率可能具有更好的检错效果, 但是, 当原始用例和衍生用例没有执行包含故障 P 的语句时, 即使实现了高覆盖率, 也无法检测到 P 故障. 随着蜕变测试在机器学习领域的广泛应用, 以语句和代码作为覆盖率的指标不再有效, 有研究人员针对特定领域设计了不同的覆盖准则. Nakajima 等人[37]认为, 针对机器学习领域, 如果输入训练的数据集是充分的, 则程序的任何执行都会覆盖所有的控制路径, 鉴于此, 他们考虑将输入域作为测试覆盖准则, 其本质是划分输入数据集, 从每一个划分区域提取具有代表性的数据, 然后形成测试输入, 该覆盖准则为蜕变关系选取提供了依据. Dwarakanath 等人[38]

也将数据集覆盖率作为选择蜕变关系以及生成衍生测试用例的指标之一.

如表 7 所示, Cao 等人[39]提出原始、衍生测试用例执行路径差距越大, 检错可能性越大; 董国伟等人[40]的研究表明, MR 原始和衍生输入变量差距个数越多, 检错率越高. 基于此, 张兴隆等人[41]提出用 MR 差异值代表故障检测程度, 该差异值体现了 MR 路径对之间的差异度和参数差异度, 从而更有效地反映 MR 检错能力. Chen 等人[42]将蜕变测试用于检测事件顺序, 考虑了原始和衍生测试用例之间的更多差异, 例如不同的事件序列和不同的输入输出变量等, 其中, 基于不同事件序列的 MR 比基于相同事件序列的 MR 具有更高的故障检测能力, 同时, 具有更丰富的输入和输出关系的 MR 具有更高的故障检测能力.

除此之外, 故障检测率 (fault detect rate, FDR) 或者变异得分 (mutation score, MS) 是评价测试用例的最直接的指标. 由于蜕变关系的有效性与原始用例和衍生用例密切相关, 该指标也被用来评价蜕变关系的检错能力.

在衡量蜕变关系有效性时, 利用变异测试技术生成程序故障, 也就是说, 在程序中添加一些错误, 得到修改后的程序, 称为变异体, 将原始与衍生测试用例依次执行变异体, 根据程序执行结果是否满足蜕变关系来检测程序错误. 其中, 变异得分是表明测试用例能否检测出变异体的量化指标, 故障检测率是检测出变异体的测试用例与总测试用例数的比值. 定义如下:

$$MS(T,r) = M_k/(M_t - M_q) \tag{4}$$

$$FDR(T,r,m) = N_f/(N_a - N_b) \tag{5}$$

其中, T 代表程序; r 是蜕变关系; M_K 是检测出的变异体数量; M_t 变异体总数; M_q 是等价变异体数量; N_f 杀死变异体 m 的测试用例数; N_a 是测试用例总数; N_b 是无效测试用例数.

上述两个指标可以准确衡量单个 MR 检错能力, 但是无法衡量不同 MR 检测能力的差异性. 变异体可能被多个 MR 检测到, 经统计, 95% 的 MR, 检测到变异体的重叠率可能高达 99%[50]. 张月[43]从变异体和测试用例角度提出了两种衡量蜕变关系差异性的衡量标准:

$$JDMR_{ij} = \frac{\left|mr_i \cup mr_j\right| - \left|mr_i \cap mr_j\right|}{\left|mr_i \cup mr_j\right|} \tag{6}$$

$$DSMR_{ij} = \sqrt{\frac{\sum (v_i^m - v_j^m)^2}{n}} \tag{7}$$

其中, mr_i 和 mr_j 分别代表蜕变关系 r_i 和 r_j 检测出变异体的测试用例集合; n 为变异体总数; v_i^m 和 v_j^m 分别代表蜕变关系 r_i 和 r_j 对变异体 m 的蜕变关系敏感度. $JDMR$ 和 $DSMR$ 的取值越大, 差异性越大. 结果表明, 该方法可以筛选掉冗余的 MR, 减少测试消耗.

上述研究均建立在蜕变关系对不同变异体检错效率一致的假设下. 然而, MR 检测程序错误具有多维性, 即对每一个变异体的检测效率存在差异. 基于此, 解晓东等人[51]将每个蜕变关系赋予一个集合属性, 该集合中保存蜕变关系对不同变异体的检测率.

上述工作在蜕变测试执行之后, 根据程序执行结果和量化指标, 对蜕变关系进行排序, 为后面测试提取信息, 这些技术归类为基于后验信息的蜕变测试. 主要观点可以总结如下: (1) MR 高覆盖率是故障检测率的既不充分也不必要条件, 针对不同的应用领域应设计不同的覆盖准则; (2) 故障检测率与测试用例执行差异、原始和衍生输入变量数量有关; (3) 利用 MR 故障检测率的度量标准来衡量蜕变关系的差异性; (4) 同一蜕变关系对不同变异体的故障检测率不同; (5) 影响蜕变关系故障检测能力的因素具有多元性.

此外, 基于先验信息的蜕变测试技术[43], 根据程序、蜕变关系特性, 在程序执行之前完成蜕变关系排序与选取, 减少执行程序带来的测试消耗. 刘利娟[44]将蜕变关系涉及的输入变量数和功能数作为划分标准, 将蜕变关系分为一元、二元和多元蜕变关系, 以及单功能和多功能蜕变关系两大类. 实验研究表明, 多功能蜕变关系一定比单功能蜕变关系的效果好, 大多数的蜕变关系的有效性与变量个数成正比, 即多元蜕变关系的有效性高于一元和二元蜕变关系, 但由于多元蜕变关系受假阴性问题的影响, 二元蜕变关系的检错效率可能高于多元蜕变关系. 基于

此, 刘利娟[44]提出了 More 蜕变关系, 减少假阴性问题的出现, 从而选择检错效率高的蜕变关系.

Spieker 等人[45]提出基于强化学习算法的自适应蜕变测试方法. 首先, 从测试套件中随机选择测试用例执行程序, 获得程序输出; 然后, 根据上下文特征选择蜕变关系, 生成衍生测试用例; 最后, 根据蜕变关系检查程序的故障, 并将程序的执行结果反馈给模型. 根据反馈信息, 例如, 从原测试用例到 MR 的映射、初始上下文特征向量、所选择的 MR 以及测试结果等, 设置适当的奖励机制, 逐渐优化蜕变关系的选择策略与选择时机, 以便用于下一次的迭代, 基于此, 实现自适应蜕变测试的过程. 该过程无需事先对全部蜕变关系排序, 测试初期, 随机选择蜕变关系执行程序, 利用程序执行信息, 实现蜕变关系的动态选择. 强化学习策略为提供有效的蜕变关系提供了有力保证.

本节主要从故障检测的角度综述了利用后验信息选取蜕变关系的研究进展, 同时分析了先验信息用于蜕变关系选择的相关工作, 有效蜕变关系具备的特性如表 7 所示.

3.2 原始测试用例生成

原始测试用例生成策略对蜕变测试故障检测能力有很大影响, 传统思路如下: 从程序输入空间内随机选取样本作为原始测试用例. 据 Segura 等人[12]统计, 随机生成原始测试用例在其统计的论文中占 57%. 为了改进随机生成测试用例的效率, 学者们早期做了以下改进, 表 8 汇总了早期提出的原始测试用例生成方法[52−58].

表 8 原始测试用例生成方法

	文献	方法	优点	缺点
早期	[52,53]	手动设置特殊值作为测试用例	克服了随机测试用例的盲目性	测试值不充分, 有效性差
	[54]	迭代蜕变测试框架, 即将衍生测试用例作为下一次测试的原始测试用例	克服了随机与特殊值测试用例的缺点	存在冗余测试用例
	[55]	利用程序路径分析, 改进迭代蜕变测试的程序输入	(1) 使用较少的测试用例覆盖更多的路径 (2) 提高了迭代测试的程序输入质量	对于大型程序复杂的循环结构, 静态分析难度较大
	[56]	使用遗传算法生成测试用例	测试用例可覆盖更多的路径	路径存在冗余
	[57]	将输入域划分为多个等价类, 基于等价类生成测试用例	具有较高故障检测率	仅适用于输入域可以划分的程序
	[58]	采用符号执行策略, 首先, 基于每个程序路径生成符号输入; 然后, 利用符号输入构造MR; 最后, 用实数替换符号输入, 生成测试用例	(1) 确保原始测试用例的高覆盖率 (2) 支持MR生成 (3) 减少测试用例数量	无法真实反映输入对程序控制流的影响
近期	[59]	动态符号执行	动态建立约束信息, 真实反映输入对程序控制流的影响	(1) 程序较小, 缺乏大程序的数据支撑 (2) 使用的测试套件不具备随机性
	[60]	自适应随机测试	有效地保证了测试用例之间的高差异性	(1) 蜕变关系需要手动构建 (2) 构建的复合蜕变关系效果较单一蜕变关系区别不大
	[61]	基于弱变异测试准则	较随机测试准则有效性高	生成的测试套件随机性差

进几年, 研究人员根据表 8 中已有的研究成果做了以下改进. 根据表 8 可知, 早期原始测试用例生成技术比较成熟的为符号执行方法, 此方法即支持蜕变关系的生成, 也确保了原始测试用例的高覆盖率. 但是该方法无法真实反映程序输入对控制流的实际影响. 基于此, Alatawi 等人[59]使用动态符号执行工具 (dynamic symbolic execution, DSE) 为蜕变测试生成原始测试用例. DSE 使用符号执行来记录输入如何影响程序中的控制流, 并通过跟踪一个具体输入执行情况, 在分支点收集和存储符号约束. 通过对这些约束的修改, 可以生成一个新的输入, 使得程序的执行朝向不同的路径. 此过程重复进行, 直到生成的测试用例满足预先设定的标准, 例如, 分支覆盖率.

根据第 3.1.2 节对蜕变关系有效性的讨论可知, 与衍生用例相距较远的原始测试用例的优先级比较高. 鉴于

此, Hui 等人[60]将自适应随机测试策略引入到原始测试用例和衍生测试用例的生成过程中, 为了尽可能选择测试覆盖率高的测试用例, 首先, 计算 k 个候选测试用例及其对应的衍生测试用例之间的距离, 然后, 选择距离最大的用例作为下一个测试用例, 若距离最大存在多个候选用例, 则计算其与已经执行的测试用例之间的距离, 选择最大的作为下一个测试用例. 该方法既保证了原始用例与衍生测试用例的高差异性, 又保证了与已执行测试用例的差异最大, 提高了测试用例的覆盖率.

此外, 随着变异测试在软件测试的广泛应用, Saha 等人[61]研究了基于弱变异测试准则的原始测试用例生成, 在该测试准则下, 测试套件至少存在一个测试用例杀死变异体. 研究表明, 与基于路径和分支覆盖准则的测试用例生成方法和随机方法相比, 弱变异覆盖准则生成原测试用例有效性明显.

本节从动态符号执行技术、自适应随机测试以及弱变异测试等, 综述了原始测试用例生成的相关研究, 近期关于原始测试用例生成方法的改进如表 8 所示.

3.3 蜕变测试过程自动化执行工具

蜕变测试的执行过程分成两个阶段. 第 1 步是根据原始测试用例和蜕变关系生成衍生测试用例, 第 2 步是将原始测试用例与衍生测试用例执行的结果进行对比, 检查是否符合蜕变关系, 从而判定程序是否存在错误. 为了提高蜕变测试执行的效率, 执行过程自动化也成为蜕变测试研究领域的重点之一. 基于此, 学者们开发了蜕变测试自动化执行工具.

早期, 关于蜕变关系自动生成, Chen 等人[24]开发了蜕变关系生成工具, MR-GEN operator, 将程序输入分类, 为每一类的输入设计相应的蜕变关系; Murphy 等人[62]实现了系统级蜕变测试工具 Amsterdam; 然而, 该工具无法确定故障位置, 只能用于缺陷检测. 鉴于此, 工具 Comedy[63]提供了一种故障定位技术, 能够自动生成被测程序的蜕变关系. 面向整个蜕变测试过程, Hong 等人[64]开发 Java 程序单元测试工具 JFuzz, 该工具将变异测试和蜕变测试技术相结合, 利用断言实现测试用例生成和测试结果分析的自动化. 对于搜索引擎和程序编译器等特定类型软件, 学者们开发了有针对性的蜕变测试工具. Zhou 等人[65]将蜕变测试应用于搜索引擎的质量检测, 但开发的工具无法涵盖影响搜索引擎性能的所有因素; Mettoc[66]是适用于编译器的蜕变测试工具, 通过生成多个具有等价关系的被测程序, 避免了向程序添加断言, 但不适用于浮点型、指针型的复杂程序.

近期, 范超[67]设计的工具能够自动识别程序的似然蜕变关系, 似然蜕变关系是指基于程序运行轨迹得到的蜕变关系启发信息, 似然蜕变关系的自动生成为构造实际蜕变关系提供参考, 该工具被应用在核电设计软件测试中, 有效地发现了似然蜕变关系. 时小芳[68]设计了包含 5 个模块的蜕变测试工具, 从原始测试用例选取、蜕变关系构造、衍生测试用例生成到程序执行、结果分析, 实现了蜕变测试过程的自动化. Spieker 等人[45]开发用于图像分析模型的测试工具 Tetraband, 该工具实现了蜕变测试的自动执行, 并允许搭建和扩展蜕变测试与其他技术的环境, 可移植性较好. 将该工具用于测试机器学习的图像分类与对象检测两个特定领域, 极大地优化了故障检测过程. Jiang 等人[69]利用蜕变测试改进自动程序修复程序, 设计了 MT-GenProg 工具, 提高了修复程序的有效性.

本节主要汇总了蜕变测试技术的相关工具, 这些工具在一定程度上实现了蜕变测试的自动化, 提高了软件测试效率, 但这些工具仍然存在如下缺陷: (1) 生成蜕变关系的类型有限, 不能使用依赖程序输出的蜕变关系或只能使用线性蜕变关系; (2) 存在静态分析的过程, 仅实现了局部自动化; (3) 难以检测复杂程序. 同时, 可移植性和可维护性是现有工具的共同缺陷, 设计通用的测试工具是未来的研究方向. 后文表 9 列出了早期和近期的蜕变测试工具[24,45,50,62−69].

4 蜕变测试应用

Segura 等人[6]汇总了截至 2016 年的蜕变测试应用领域, 包括: 网络服务与应用、计算机图形学、仿真与建模、嵌入系统等在内的 12 个领域. 近几年, 随着人工智能的发展, 机器学习已经变得越来越流行, 并广泛应用于数据挖掘、模式识别、生物信息学、自然语言处理、机器视觉等领域. Oracle 问题的存在同样使得这类软件可靠性难以

保证. 蜕变测试为这些领域的软件测试提供了可行途径. 除了解决测试过程中 Oracle 问题以外, 蜕变测试还用于评估软件质量特征[10]. 表 10 汇总了近 5 年蜕变测试应用领域, 以下是不同领域的具体介绍[3,4,6−8,28−30,44,48,49,57,69−100].

表 9 蜕变测试工具

文献	工具	开发语言	功能
早期	MR-GEN operator[24]	Java	蜕变关系生成
	KABU[50]	Java	
	Comedy[63]	Java	蜕变测试过程
	JFuzz[64]	Java	
	搜索引擎[65]	JavaScript	特定软件蜕变测试
	Mettoc[66]	C语言	
	Amsterdam[62]	Python	
近期	似然蜕变关系发现工具[67]	Java	蜕变关系生成
	南华大学蜕变测试软件[68]		蜕变测试过程
	Tetraband[45]		特定软件蜕变测试
	MT-GenProg[69]		

表 10 应用领域

应用领域	程序类型	解决问题
机器学习	监督学习程序[3,71,72,82]	Oracle问题
	深度学习模型[49,73,76−78,84−88]	
	无监督学习程序[70]	评估软件质量
	机器翻译系统[7,74]	数据集质量评估
密码学	加密算法[4]	Oracle问题
编译器	GCC编译器[78]	Oracle问题
	SNL编译器[44]	
专用软件	生物信息学软件[6,89]	Oracle问题
	搜索系统[28−30,90,91]	
	宇航局GMSEC软件总线[92]	
	导航系统[57,93,94]	
	业务流程软件系统[44]	
	SWMM软件[95]	
	订单生成系统[96]	
	数据分析软件[97]	评估软件质量
	自动修复程序技术[69]	
	内存系统[98]	
	大数据应用程序Dbpedia2[8]	
	世界地图软件OpenStreetMap[99]	
	问答系统[100]	
其他	约束求解器[79]	Oracle问题
	调试器[80]	
	测试用例生成[78]	
	仿真模型[81]	评估软件质量

(1) 机器学习系统

对于无监督聚类程序, Yang 等人[70]假设了一套蜕变关系, 以加强对程序的更好理解和使用. 为了解决测试线

性分类算法过程中的 Oracle 问题, Yang 等人[71]将线性分类算法的基本属性转化成过去执行依赖 MR 的形式, 并将其运用到 9 个著名的线性分类算法中, 实现了较强的故障检测效率.

近期, 蜕变测试还用于模式识别、自然语言处理和自动化系统等场景. Xiu 等人[72]将蜕变测试技术用于包含隐藏视觉标记的图像分类问题, 利用图像的分离和遮挡特性, 生成 MR 调整机器学习系统的输入和输出, 从而构建更精确的二元分类模型. 当图像中添加对抗性攻击, 会导致深度学习模型错误分类. 为此, Mekala 等人[73]使用蜕变测试来检测对抗性攻击, 该方法可以检测出图像改变非常小、人肉眼无法检测的操作. 为了缓解图像分类程序的 Oracle 问题, 刘佳洛等人[3]提出一种适用图像分类程序的蜕变测试框架.

不同类型文本消息的机器翻译是自然语言处理领域常见的应用场景, 其输入数据质量至关重要. Yan 等人[7]选择对某一电影的评论作为数据集, 利用蜕变测试技术验证和评估该输入数据集质量. 结果表明了该方法可以有效地识别翻译不正确的语句. 神经机器翻译模型对鲁棒性要求较高. 为了研究该问题, 钟文康等人[74]利用不同语言粒度对翻译结果造成的影响, 结合蜕变测试技术, 达到了不借助参考译文也能评估翻译质量的目的. 该方法运用在不同的翻译系统中, 极大地提高了原有方法的皮尔逊相关系数与斯皮尔曼相关系数. 因具备实时响应用户和交互界面简单等优点, 聊天机器人越来越受到人们的欢迎. 为了克服其输入与输出数据难以预测的问题, Božić[75]将特定领域知识概念化, 应用蜕变测试技术, 对聊天机器人程序进行功能测试.

为了解决自动驾驶系统中的安全问题, Zhang 等人[76]提出了一种融入蜕变测试技术的无监督自动验证框架, 用来检查这些系统的一致性. Zhou 等人[77]提出了一种将蜕变测试与模糊处理相结合的测试策略, 并阐述了如何使用该策略检测百度阿波罗自动驾驶软件中的致命错误. 除此之外, 无人机技术得到广泛应用, 但是其测试可行性较低, 测试难度较大, Lindvall 等人[78]将蜕变测试原理与基于模型的测试相结合, 开发了一个模拟自主无人机系统的自动化测试框架, 该框架降低了无人机测试代价与难度.

(2) 加密算法

由于加密算法的复杂性与紧凑性, 极大增加了其检测的难度. Pugh 等人[4]通过加密算法特有的编码规范来构造蜕变关系, 并设计测试用例实现对输入空间的系统覆盖. 结果表明, 这种方法能够检测复杂加密程序中的错误.

(3) 编译器

编译器是将高级程序语言转化成目标代码, 保证编译器正确性对程序执行非常重要. 但编译器的测试同样存在 Oracle 问题, 刘利娟[44]将蜕变测试应用于 SNL 编译器. 在将蜕变测试应用于编译器测试的过程中, 生成测试用例集合也是需要考虑的重要因素. 为此, 王丽瑶[79]将测试用例生成过程自动化, 设计并开发了相应的支持工具, 提高测试效率.

(4) 其他

Akgün 等人[80]通过比较同一约束的两种不同实现, 将蜕变测试用于约束求解器. 考虑到调节器的输入由源代码和一系列调试操作组成, Tolksdorf 等人[81]提出了交互式蜕变测试. 该方法是在程序执行的过程中, 确定输出之间的关系. 实验证明, 该测试方法检测出了之前没有检出的错误. Olsen 等人[82]通过建立基于参数和行为之间的蜕变关系来测试基于代理和离散事件的仿真模型.

5 并行程序蜕变测试

5.1 并行程序实施

并行程序是并行软件的主要组成部分, 该类程序由多个同时执行的流程组成, 在执行过程中交互信息, 共同实现整个程序的功能.

根据编程模式分类, 并行程序分为 4 类[101]. (1) 主从式编程: 在多个流程中, 设定一个主流程和多个从流程, 其中主流程向从流程发送计算数据, 并接收来自从流程的反馈结果; 从流程用于完成具体的计算任务, 并将结果发送到主流程. (2) 单程序多数据流: 在该模式中, 所有流程的代码相同, 但是计算不同的数据, 最终计算结果被汇集起来. (3) 数据流水线: 每个流程完成程序特定功能的子任务, 流程之间具有一定的顺序, 交互仅存在相邻流程之间. (4) 分治策略: 将大问题分解成多个性质相同的子问题, 直到子问题变得容易解决; 最后, 将各个子问题的解汇集成

原问题的解.

常见的并行程序开发方法主要有 3 种[102]: (1) 基于并行编译系统自动实现串行代码并行化; (2) 使用 Ada、Java、Acala 等常见的并行设计语言; (3) 使用外部机制实现并行, 常用的有, 消息传递机制 (MPI、PVM), 共享变量机制 (OpenMP) 等. 基于程序不同的存储机制, 并行程序分为两类: (1) 分布存储: 每个流程拥有独立的存储器, 流程之间通过传递消息实现交互; (2) 共享存储: 除了独立的存储器之外, 还存在共享存储单元, 流程之间通过对共享存储单元的存取实现交互. 图 1 是共享存储的示例程序, 其中, *f*1 与 *f*2 并行执行, 共享存储器 *Buffer* 的长度为 2, 通过对共享存储器 *Buffer* 的存取实现交互. 图 2 为分布存储并行程序的例子, *f*1 与 *f*2 通过通信实现并行执行, 通信变量为 *R*1 和 *R*2.

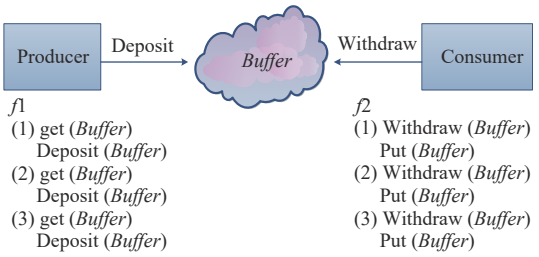


图 1 共享存储并行程序例子

```
f1:
1: int R1=5;
2: message-send (f2, R1);
3: message-receive (R2, f2);
4: if (R1>R2){
5:     max=R1;}
   else{
6:     max=R2;}
```

```
f2:
1: int R2=10;
2: message-send (f1, R2);
3: message-receive (R1, f1);
4: if (R1>R2){
5:     min=R2;}
   else{
6:     min=R1;}
```

图 2 分布存储并行程序例子

5.2 可能的研究思路

良好的可移植性、强大的功能和高效的执行效率使得并行程序在工业生产、科学计算等领域得到了广泛应用. 由于多流程并行, 以及流程交互顺序的不同, 导致并行程序执行具有一定的不确定性, 相同输入下的不同运行, 将生成不同的程序执行轨迹或输出结果. 由于并行程序的并发性、不确定性以及同步等, 使得程序易发生死锁、数据竞争等与交互相关的缺陷, 并行程序的测试引起众多学者的关注. 已有工作中, 并行程序测试主要集中在死锁、数据竞争、原子性违背和顺序违背等缺陷的检测, 如何运用蜕变测试技术解决并行程序 Oracle 问题, 鲜有报道. 本节基于已有的蜕变测试工作, 讨论并展望并行程序蜕变测试可能的研究主题和内容.

(1) 蜕变关系是蜕变测试的核心. 为了构造蜕变关系, 虽然学者们做出了大量的研究, 有复合蜕变关系生成、基于机器学习的蜕变关系构建等, 但是, 由于没有考虑程序的特点, 已有方法难以直接用于并行程序的蜕变测试. 针对并行程序并行、交互和不确定执行等, 拓展已有蜕变关系的生成策略, 是一种可行途径.

分析程序输入与交互顺序对并行程序的影响, 结合“对称”蜕变关系模式的思想[25], 分别针对程序计算, 以及交互相关缺陷检测构造蜕变关系. 针对分布存储并行程序, 考虑每个流程实现的功能设计局部蜕变关系. 采用基于搜索的方法寻找满足一定条件, 例如, 高故障检测率和高覆盖率等的蜕变关系及其多个流程蜕变关系的组合. 为此, 可以采用一组关系序列来表示多个流程蜕变关系, 从而寻找合适的序列逐步构造检测范围更广的蜕变关系. 进一步, 根据“蜕变关系输出模式”[35]的思想, 将流程蜕变关系和多个流程相关蜕变关系作为不同集合, 利用集合操作, 分析集合之间关系, 生成全局蜕变关系. 综合利用蜕变关系能够区分计算和通信相关的缺陷, 同时, 也可以实现缺陷的定位. 类似地, 基于上述思想, 针对共享存储并行程序, 从共享变量的状态改变, 以及流程对共享存储单元的存取顺序等方面入手, 构造蜕变关系; 针对实时并发系统的实时性要求, 根据任务的执行时间、任务的执行顺序、共享资源的使用, 以及环境行为的改变等信息构造蜕变关系, 从而实现对实时系统的检测.

除此之外, 运用机器学习方法预测并行程序蜕变关系时, 需要在考虑程序控制流图的路径、节点等特征的基础上, 提取并行程序的相关特征, 例如, 并行程序的通信顺序、共享变量的存取顺序、各个流程内部的依赖关系、流程局部功能、流程相关输入变量等知识. 这些特征的提取可能还需反映程序某一侧面的表示机制, 例如通信图和存取顺序图等.

(2) 蜕变关系选择的主要标准是高覆盖率、高故障检测率和原始衍生测试用例之间的高差异度. 在并行程序

的蜕变关系选择中, 根据全局和局部蜕变关系之间的相关性, 对存在属于关系和交集关系的蜕变关系进行约简, 将全局覆盖与局部覆盖相结合, 最终实现程序计算与并发缺陷的全覆盖. 此外, 已有研究表明, 原始和衍生测试用例之间差异性越大, 蜕变关系的检错率越高[37]. 并行程序的测试用例需要考虑程序输入和流程交互顺序两部分. 因此, 在比较蜕变关系时, 既要考虑程序输入的高差异性, 也要考虑流程交互的高差异性.

已有大多蜕变关系的选择采用后验信息模式, 即在测试之前执行程序获得相关信息. 但是, 当程序规模较大时, 后验信息的获取代价也会较高, 这在并行程序测试中尤其明显. 因为执行过程的不确定性将给信息提取和利用带来不可忽视的额外代价. 鉴于此, 进一步研究并行程序特性、蜕变关系、原始和衍生测试用例、通信变量、共享变量、任务的执行时间、任务执行顺序对程序流程的影响, 与蜕变测试效率之间的关系, 例如, 根据蜕变关系引起的每个流程相关输入的差异性, 以及不同流程蜕变关系之间的相关性, 在程序运行之前进行蜕变关系排序, 降低蜕变测试的执行代价.

此外, 将蜕变测试与人工智能技术相结合, 是提高蜕变测试效率的有效方法. 首先, 根据先验信息提取有效特征, 例如, 每个流程相关输入的差异性; 然后, 选择合适的机器学习算法, 将处理可能出现的数据不平衡问题, 实现蜕变关系质量自动化预测; 利用强化学习算法的优势, 结合并行程序上下文特征可能出现的不确定性, 选择合适的蜕变关系, 根据程序输出结果, 设置不同流程的优先级, 结合已有的奖励机制, 优化选择策略, 从而选择符合预定标准的蜕变关系.

(3) 原始测试用例对蜕变测试性能有着重要影响. 并行程序的测试不仅需要程序输入, 还要考虑交互顺序. 在并行程序中, 给定相同程序输入, 程序多次运行可能得到不同的执行轨迹. 考虑流程交互对程序执行的影响, 选择更容易产生高差异度输入用例的流程交互顺序, 改进动态符号执行方法, 生成覆盖所有分支的测试用例. 进一步, 针对流程之间的冗余交互, 制定约简策略, 在简化问题规模的前提下, 实现所有流程交互的覆盖.

根据第 2.2 节的讨论, 基于弱变异测试准则的测试用例生成效率优于随机方法. 在对并行程序进行变异时, 变异算子将分为两种: 传统变异算子和并行变异算子. 考虑并行变异算子引起通信环境或者共享变量状态的改变, 基于两种变异算子构造变异分支, 采用启发式算法生成覆盖所有分支的测试用例, 实现分支全覆盖, 从而提高错误被发现的概率, 使得基于这些原始测试用例, 实施蜕变测试时, 不仅能够检测计算缺陷, 还可以发现与共享存储或通信相关的缺陷.

(4) 根据第 2.5 节对蜕变工具的描述可知, 已有蜕变工具没有关注并行程序特性. 考虑并行、交互、不确定性和同步等是并行程序蜕变测试工具首先要考虑的问题. 此外, 不管是串行程序还是并行程序, 在原始测试用例选取、蜕变关系构造、衍生测试用例生成、程序执行和结果分析等阶段, 并行技术是优化蜕变测试过程, 改善测试效率的重要手段. 利用程序的并行和分布性, 同时产生不同流程的局部蜕变关系和测试用例, 进一步, 并行产生于多个流程相关的全局复合蜕变关系. 此外, 根据蜕变测试工具支持的最大资源和被测程序需求的最少资源, 如何利用剩余资源改善蜕变测试过程, 是一个值得研究的问题.

(5) 蜕变测试与其他技术相结合, 是提高并行程序测试效率的另一有效手段.

将蜕变测试与变异测试相结合. 由于并行程序不确定性的存在, 被测程序与变异体输出结果存在不确定性, 分析程序功能与交互序列对程序输出的影响, 借助蜕变测试输入-输出关系的思想, 提出比较程序输出结果的合理方法, 以便准确观察变异体与原始程序之间的行为差异.

将蜕变测试与覆盖测试相结合. 已有工作多将覆盖测试用于前期蜕变测试, 生成原始用例. 类似地, 一方面, 在利用局部蜕变关系, 针对某个流程实施蜕变测试时, 仍然能够将分支、路径等覆盖测试准则用于蜕变关系的选取和原始测试用例生成. 另一方面, 对并行程序而言, 流程之间的交互是测试的关键之一. 因此, 利用覆盖测试生成原始测试用例时, 在考虑传统覆盖准则的前提下, 结合其他测试技术, 例如可达性测试, 是需要考虑的重点. 此外, 综合考虑被测程序的执行轨迹和蜕变关系的满足情况, 有利于缺陷定位, 在并行程序测试时, 分析局部流程和全局蜕变关系之间的相关性, 是区分计算缺陷和交互缺陷的一种可能的解决方案.

利用搜索方法改进蜕变测试过程, 已经得到了有效的验证, 例如, 将启发式优化算法用于复合蜕变关系的生成[21]. 缩小搜索空间是将启发式方法用于蜕变测试时, 提高测试效率的途径之一. 并行程序受不确定性等因素的

影响, 比串行程序更复杂, 相应地, 问题搜索空间更大. 利用测试之前和测试过程的知识, 缩小问题的搜索空间, 解决流程交互顺序产生, 以及局部功能的蜕变关系生成, 是并行程序蜕变测试值得研究的另一主题.

综上所述, 表 11 列出了并行程序蜕变测试的可能研究主题, 图 3 展示了该部分的示意图.

表 11 并行程序蜕变测试可能的研究主题

方面	主题
蜕变关系生成	分别检测计算和交互缺陷的蜕变关系生成 单流程变关系和多流程相关蜕变关系的生成 不同类型并行程序的蜕变关系生成 用于机器学习预测的特征提取和预测方法的选择
蜕变关系选取	并行程序蜕变关系有效性的评价指标 基于机器学习的蜕变关系有效性预测
原始测试用例生成	面向并行覆盖准则的原始测试用例生成
蜕变测试工具	基于并行程序特性的蜕变测试工具改进 利用并行技术优化蜕变测试过程 测试资源的合理分配
与其他技术结合	利用其他测试技术用于改进蜕变测试过程 利用蜕变测试技术改进其他测试技术

图 3 并行程序蜕变测试可能研究思路的示意图

6 结束语

本文从基本概念、过程优化和应用领域 3 个模块综述了蜕变测试的研究进展, 重点是近 5 年国内外的研究工

作较早期成果的改进. 其中, 基本概念模块主要包括蜕变测试原理和蜕变测试过程; 过程优化模块包括: 蜕变关系、原始测试用例生成、蜕变测试过程, 以及蜕变测试工具 4 个子模块; 应用领域模块主要阐述了近 5 年蜕变测试的应用领域.

随着并行程序被广泛应用, 并行程序测试逐渐引起学者们的关注. 本文基于已有的蜕变测试研究工作, 结合并行程序并行、不确定性以及交互等特征, 从蜕变关系生成、蜕变关系选取、原始测试用例生成、蜕变测试工具以及与其他技术结合等 5 个模块, 讨论了并行程序蜕变测试技术可能的研究主题与思路.

虽然现有的蜕变测试工作已经取得了很大进展. 开发自动化蜕变测试工具, 在大型软件开发过程中充分发挥蜕变测试的优势, 将蜕变测试更好地用于工业软件测试仍然是非常值得关注的问题.

References:

[1] Chen TY, Cheung SC, Yiu SM. Metamorphic testing: A new approach for generating next test cases. Technical Report HKUST-CS98-01, Hong Kong: Hong Kong University of Science and Technology, 1998.

[2] Segura S, Towey D, Zhou ZQ, Chen TY. Metamorphic testing: Testing the untestable. IEEE Software, 2018, 37(3): 46–53. [doi: 10.1109/MS.2018.2875968]

[3] Liu JL, Yao Y, Huang S, Hui ZW, Chen Q, Kou DL, Zhang ZW. Metamorphic testing framework for machine learning image classification program. Computer Engineering and Applications, 2020, 56(17): 69–77 (in Chinese with English abstract). [doi: 10.3778/j.issn.1002-8331.1906-0242]

[4] Pugh S, Raunak MS, Kuhn DR, Kacker R. Systematic testing of post-quantum cryptographic implementations using metamorphic testing. In: Proc. of the 4th IEEE/ACM Int'l Workshop on Metamorphic Testing. Montreal: IEEE, 2019. 2–8. [doi: 10.1109/MET.2019.00009]

[5] Donaldson AF, Lascu A. Metamorphic testing for (graphics) compilers. In: Proc. of the 1st Int'l Workshop on Metamorphic Testing. Austin: ACM, 2016. 44–47. [doi: 10.1145/2896971.2896978]

[6] Shahri MP, Srinivasan M, Reynolds G, Bimczok D, Kahanda I, Kanewala U. Metamorphic testing for quality assurance of protein function prediction tools. In: Proc. of the 2019 IEEE Int'l Conf. on Artificial Intelligence Testing. Newark: IEEE, 2019. 140–148. [doi: 10.1109/AITest.2019.00017]

[7] Yan BY, Yecies B, Zhou ZQ. Metamorphic relations for data validation: A case study of translated text messages. In: Proc. of the 4th IEEE/ACM Int'l Workshop on Metamorphic Testing. Montreal: IEEE, 2019. 70–75. [doi: 10.1109/MET.2019.00018]

[8] Auer F, Felderer M. Addressing data quality problems with metamorphic data relations. In: Proc. of the 4th IEEE/ACM Int'l Workshop on Metamorphic Testing. Montreal: IEEE, 2019. 76–83. [doi: 10.1109/MET.2019.00019]

[9] Chen TY, Kuo FC, Towey D, Zhou ZQ. A revisit of three studies related to random testing. Science China Information Sciences, 2015, 58(5): 1–9. [doi: 10.1007/s11432-015-5314-x]

[10] Zhou ZQ, Xiang SW, Chen TY. Metamorphic testing for software quality assessment: A study of search engines. IEEE Trans. on Software Engineering, 2016, 42(3): 264–284. [doi: 10.1109/TSE.2015.2478001]

[11] Dong GW, Xu BW, Chen L, Nie CH, Wang LL. Survey of metamorphic testing. Journal of Frontiers of Computer Science and Technology, 2009, 3(2): 130–143 (in Chinese with English abstract). [doi: 10.3778/j.issn.1673-9418.2009.02.002]

[12] Segura S, Fraser G, Sanchez AB, Ruiz-Cortés A. A survey on metamorphic testing. IEEE Trans. on Software Engineering, 2016, 42(9): 805–824. [doi: 10.1109/TSE.2016.2532875]

[13] Chen TY, Kuo FC, Liu H, Poon PL, Towey D, Tse TH. Metamorphic testing: A review of challenges and opportunities. ACM Computing Surveys, 2019, 51(1): 4. [doi: 10.1145/3143561]

[14] Hui ZW, Huang S, Li H, Liu JH. Formal description and decomposition of metamorphic relation. Computer Engineering and Design, 2016, 37(2): 405–412 (in Chinese with English abstract). [doi: 10.16208/j.issn1000-7024.2016.02.024]

[15] Segura R, Durán A, Troya J, Cortés AR. A template-based approach to describing metamorphic relations. In: Proc. of the 2nd IEEE/ACM Int'l Workshop on Metamorphic Testing. Buenos Aires: IEEE, 2017. 3–9. [doi: 10.1109/MET.2017.3]

[16] Kanewala U, Bieman JM. Using machine learning techniques to detect metamorphic relations for programs without test oracles. In: Proc. of the 24th IEEE Int'l Symp. on Software Reliability Engineering. Pasadena: IEEE, 2013. 1–10. [doi: 10.1109/ISSRE.2013.6698899]

[17] Kanewala U. Techniques for automatic detection of metamorphic relations. In: Proc. of the 7th IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. Cleveland: IEEE, 2014. 237–238. [doi: 10.1109/ICSTW.2014.62]

[18] Liu H, Liu X, Chen TY. A new method for constructing metamorphic relations. In: Proc. of the 12th Int'l Conf. on Quality Software (QSIC). Xi'an: IEEE, 2012. 59–68. [doi: 10.1109/QSIC.2012.10]

[19] Dong GW, Xu BW, Chen TY, Nie CH, Wang LL. Case studies on testing with compositional metamorphic relations. Journal of Southeast University, 2008, 24(4): 437–443.

[20] Zhang J, Chen JJ, Hao D, Xiong YF, Xie B, Zhang L, Mei H. Search-based inference of polynomial metamorphic relations. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 701–712. [doi: 10.1145/2642937.2642994]

[21] Goffi A, Gorla A, Mattavelli A, Pezzè M, Tonella P. Search-based synthesis of equivalent method sequences. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Hong Kong: ACM, 2014. 366–376. [doi: 10.1145/2635868.2635888]

[22] Goffi A. Automatic generation of cost-effective test oracles. In: Proc. of the 36th Int'l Conf. on Software Engineering. Hyderabad: ACM, 2014. 678–681. [doi: 10.1145/2591062.2591078]

[23] Su FH, Bell J, Murphy C, Kaiser G. Dynamic inference of likely metamorphic properties to support differential testing. In: Proc. of the 10th IEEE/ACM Int'l Workshop on Automation of Software Test. Florence: IEEE, 2015. 55–59. [doi: 10.1109/AST.2015.19]

[24] Chen TY, Poon PL, Xie XD. METRIC: Metamorphic relation identification based on the category-choice framework. Journal of Systems and Software, 2016, 116: 177–190. [doi: 10.1016/j.jss.2015.07.037]

[25] Zhang XL, Yu L, Hou XM, Hou SF. Method of metamorphic relations constructing for object-oriented software testing. Computer Science, 2017, 44(S2): 485–489, 515 (in Chinese with English abstract).

[26] Hou XM, Yu L, Zhang XL, Li ZB. Constructing method of metamorphic relations in object-oriented software testing. Journal of Computer Applications, 2015, 35(10): 2990–2994 (in Chinese with English abstract).

[27] Zhou ZQ, Zhang SJ, Hagenbuchner M, Tse TH, Kuo FC, Chen TY. Automated functional testing of online search services. Software:Testing, Verification and Reliability, 2012, 22(4): 221–243. [doi: 10.1002/stvr.437]

[28] Segura S, Parejo JA, Troya J, Ruiz-Cortés A. Metamorphic testing of RESTful web APIs. IEEE Trans. on Software Engineering, 2018, 44(11): 1083–1099. [doi: 10.1109/TSE.2017.2764464]

[29] Zhou ZQ, Sun LQ, Chen TY, Towey D. Metamorphic relations for enhancing system understanding and use. IEEE Trans. on Software Engineering, 2020, 46(10): 1120–1154. [doi: 10.1109/TSE.2018.2876433]

[30] Wu CH, Sun LQ, Zhou ZQ. The impact of a dot: Case studies of a noise metamorphic relation pattern. In: Proc. of the 4th IEEE/ACM Int'l Workshop on Metamorphic Testing. Montreal: IEEE, 2019. 17–23. [doi: 10.1109/MET.2019.00011]

[31] Kanewala U, Bieman JM, Ben-Hur A. Predicting metamorphic relations for testing scientific software: A machine learning approach using graph kernels. Software: Testing, Verification and Reliability, 2016, 26(3): 245–269. [doi: 10.1002/stvr.1594]

[32] Zhang PC, Zhou XW, Pelliccione P, Leung H. RBF-MLMR: A multi-label metamorphic relation prediction approach using RBF neural network. IEEE Access, 2017, 5: 21791–21805. [doi: 10.1109/ACCESS.2017.2758790]

[33] Mu XY, Fan Y, Li SJ, Zhang P, Liu L. Method of generating metamorphic relationship based on gradient descent algorithm. Journal of Jilin University (Science Edition), 2020, 58(6): 1429–1435 (in Chinese with English abstract). [doi: 10.13413/j.cnki.jdxblxb.2019456]

[34] Xiang ZL, Wu HR, Yu F. A genetic algorithm-based approach for composite metamorphic relations construction. Information, 2019, 10(12): 392. [doi: 10.3390/info10120392]

[35] Qiu K, Zheng Z, Chen T, Poon PL. Theoretical and empirical analyses of the effectiveness of metamorphic relation composition. IEEE Trans. on Software Engineering, 2020, 48(3): 1001–1017. [doi: 10.1109/TSE.2020.3009698]

[36] Asrafi M, Liu H, Kuo FC. On testing effectiveness of metamorphic relations: A case study. In: Proc. of the 5th Int'l Conf. on Secure Software Integration and Reliability Improvement. Jeju: IEEE, 2011. 147–156. [doi: 10.1109/SSIRI.2011.21]

[37] Nakajima S, Bui HN. Dataset coverage for testing machine learning computer programs. In: Proc. of the 23rd Asia-Pacific Software Engineering Conf. . Hamilton: IEEE, 2016. 297–304. [doi: 10.1109/APSEC.2016.049]

[38] Dwarakanath A, Ahuja M, Podder S, Vinu S, Naskar A, Koushik MV. Metamorphic testing of a deep learning based forecaster. In: Proc. of the 4th IEEE/ACM Int'l Workshop on Metamorphic Testing. Montreal: IEEE, 2019. 40–47. [doi: 10.1109/MET.2019.00014]

[39] Cao YX, Zhou ZQ, Chen TY. On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions. In: Proc. of the 13th Int'l Conf. on Quality Software. Nanjing: IEEE, 2013. 153–162. [doi: 10.1109/QSIC.2013.43]

[40] Dong GW, Nie CH, Xu BW. Effectively metamorphic testing based on program path analysis. Chinese Journal of Computers, 2009, 32(5): 1002–1013 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2009.01002]

[41] Zhang XL, Yu L, Hou XM, Li ZB, Li G. Method for selection of metamorphic relations based on differences analysis. Computer Engineering and Design, 2017, 38(1): 103–109 (in Chinese with English abstract). [doi: 10.16208/j.issn1000-7024.2017.01.020]

[42] Chen J, Wang YL, Guo Y, Jiang MY. A metamorphic testing approach for event sequences. PLoS One, 2019, 14(2): e0212476. [doi: 10.1371/journal.pone.0212476]

[43] Zhang Y. Research on numerical program metamorphic relations construction and optimization [MS. Thesis]. Xiamen: Huaqiao University, 2019 (in Chinese with English abstract).

[44] Liu LJ. Study on effectiveness of metamorphic relation [MS. Thesis]. Changchun: Jinlin University, 2018 (in Chinese with English abstract).

[45] Spieker H, Gotlieb A. Adaptive metamorphic testing with contextual bandits. Journal of Systems and Software, 2020, 165: 110574. [doi: 10.1016/j.jss.2020.110574]

[46] Chen TY, Huang DH, Tse TH, Zhou ZQ. Case studies on the selection of useful relations in metamorphic testing. In: Proc. of the 4th Ibero-American Symp. on Software Engineering and Knowledge Engineering. Madrid: IEEE, 2004. 569–583.

[47] Just R, Schweiggert F. Automating software tests with partial oracles in integrated environments. In: Proc. of the 5th Workshop on Automation of Software Test. Cape Town: ACM, 2010. 91–94. [doi: 10.1145/1808266.1808280]

[48] Just R, Schweiggert F. Automating unit and integration testing with partial oracles. Software Quality Journal, 2011, 19(4): 753–769. [doi: 10.1007/s11219-011-9151-x]

[49] Wang R, Ben KR. Researches on basic criterion and strategy of constructing metamorphic relations. Computer Science, 2012, 39(1): 115–119 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-137X.2012.01.026]

[50] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5): 649–678. [doi: 10.1109/TSE.2010.62]

[51] Xie XD, Peng SM, Liu Y, Wang KW, Wang T, Wang C. Sensitivity of metamorphic relationships and its cluster analysis. Acta Electronica Sinica, 2016, 44(5): 1196–1201 (in Chinese with English abstract). [doi: 10.3969/j.issn.0372-2112.2016.05.026]

[52] Chen TY, Kuo FC, Liu Y, Tang A. Metamorphic testing and testing with special values. In: Proc. of the 5th Int'l Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. Beijing: ACIS, 2004. 128–134.

[53] Wu P, Shi XC, Tang JJ, Lin HM, Chen TY. Metamorphic testing and special case testing: A case study. Ruan Jian Xue Bao/Journal of Software, 2005, 16(7): 1210–1220 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/1210.htm [doi: 10.1360/jos161210]

[54] Wu P. Iterative metamorphic testing. In: Proc. of the 29th Annual Int'l Computer Software and Applications Conf. Edinburgh: IEEE, 2005. 19–24. [doi: 10.1109/COMPSAC.2005.93]

[55] Dong GW, Nie CH, Xu BW, Wang LL. An effective iterative metamorphic testing algorithm based on program path analysis. In: Proc. of the 7th Int'l Conf. on Quality Software. Portland: IEEE, 2007. 292–297. [doi: 10.1109/QSIC.2007.4385510]

[56] Batra G, Sengupta J. 2011. An efficient metamorphic testing technique using genetic algorithm. In: Proc. of the 5th Int'l Conf. on Information Intelligence, Systems, Technology and Management. Gurgaon: Springer, 2011. 180–188. [doi: 10.1007/978-3-642-19423-8_19]

[57] Chen LL, Cai LZ, Liu J, Liu ZY, Wei SY, Liu P. An optimized method for generating cases of metamorphic testing. In: Proc. of the 6th Int'l Conf. on New Trends in Information Science, Service Science and Data Mining. Taipei: IEEE, 2012. 439–443.

[58] Dong GW, Guo T, Zhang PH. Security assurance with program path analysis and metamorphic testing. In: Proc. of the IEEE 4th Int'l Conf. on Software Engineering and Service Science. Beijing: IEEE, 2013. 193–197. [doi: 10.1109/ICSESS.2013.6615286]

[59] Alatawi E, Miller L, Søndergaard H. Generating source inputs for metamorphic testing using dynamic symbolic execution. In: Proc. of the 1st Int'l Workshop on Metamorphic Testing. Austin: ACM, 2016. 19–25. [doi: 10.1145/2896971.2896980]

[60] Hui ZW, Huang S, Chua C, Chen TY. Semiautomated metamorphic testing approach for geographic information systems: An empirical study. IEEE Trans. on Reliability, 2020, 69(2): 657–673. [doi: 10.1109/TR.2019.2931561]

[61] Saha P, Kanewala U. Fault detection effectiveness of source test case generation strategies for metamorphic testing. In: Proc. of the 3rd Int'l Workshop on Metamorphic Testing. Gothenburg: ACM, 2018. 2–9. [doi: 10.1145/3193977.3193982]

[62] Murphy C, Shen K, Kaiser G. Automatic system testing of programs without test oracles. In: Proc. of the 18th Int'l Symp. on Software Testing and Analysis. Chicago: ACM, 2009. 189–200. [doi: 10.1145/1572272.1572295]

[63] Jin H, Jiang YY, Liu N, Xu C, Ma XX, Lu J. Concolic metamorphic debugging. In: Proc. of the 39th IEEE Annual Computer Software and Applications Conf. Taichung: IEEE, 2015. 232–241. [doi: 10.1109/COMPSAC.2015.79]

[64] Zhu H. JFuzz: A tool for automated java unit testing based on data mutation and metamorphic testing methods. In: Proc. of the 2nd Int'l Conf. on Trustworthy Systems and Their Applications. Hualien: IEEE, 2015. 8–15. [doi: 10.1109/TSA.2015.13]

[65] Zhou ZQ, Tse TH, Kuo FC, Chen TY. Automated functional testing of Web search engines in the absence of an oracle. Technical Report TR-2007-06, Hong Kong: The University of Hong Kong, 2007.

[66] Tao QM, Wu W, Zhao C, Shen WW. An automatic testing approach for compiler based on metamorphic testing technique. In: Proc. of the 2010 Asia Pacific Software Engineering Conf. Sydney: IEEE, 2010. 270–279. [doi: 10.1109/APSEC.2010.39]

[67] Fan C. The design and application of dynamic discovery tool for likely metamorphosis relation [MS. Thesis]. Hengyang: University of South China, 2018 (in Chinese with English abstract).

[68] Shi XF. Design and implementation of metamorphic testing tool based on linear metamorphic relations [MS. Thesis]. Hengyang: University of South China, 2018 (in Chinese with English abstract).

[69] Jiang MY, Chen TY, Kuo FC, Towey D, Ding ZH. A metamorphic testing approach for supporting program repair without the need for a test oracle. Journal of Systems and Software, 2017, 126: 127–140. [doi: 10.1016/j.jss.2016.04.002]

[70] Yang S, Towey D, Zhou ZQ. Metamorphic exploration of an unsupervised clustering program. In: Proc. of the 4th IEEE/ACM Int'l Workshop on Metamorphic Testing. Montreal: ACM, 2019. 48–54. [doi: 10.1109/MET.2019.00015]

[71] Yang YZ, Li ZA, Wang HY, Xu C, Ma XX. Towards effective metamorphic testing by algorithm stability for linear classification programs. Journal of Systems and Software, 2021, 180: 111012. [doi: 10.1016/j.jss.2021.111012]

[72] Xu LM, Towey D, French AP, Benford S, Zhou ZQ, Chen TY. Enhancing supervised classifications with metamorphic relations. In: Proc. of the 3rd IEEE/ACM Int'l Workshop on Metamorphic Testing. Gothenburg: IEEE, 2018. 46–53.

[73] Mekala RR, Magnusson GE, Porter A, Lindvall M, Diep M. Metamorphic detection of adversarial examples in deep learning models with affine transformations. In: Proc. of the 4th IEEE/ACM Int'l Workshop on Metamorphic Testing. Montreal: IEEE, 2019. 55–62. [doi: 10.1109/MET.2019.00016]

[74] Zhong WK, Ge JD, Chen X, Li CY, Tang Z, Luo B. Multi-granularity metamorphic testing for neural machine translation system. Ruan Jian Xue Bao/Journal of Software, 2021, 32(4): 1051−1066 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6221.htm [doi: 10.13328/j.cnki.jos.006221]

[75] Božić J. Ontology-based metamorphic testing for chatbots. Software Quality Journal, 2022, 30(3): 227–251. [doi: 10.1007/s11219-020-09544-9]

[76] Zhang MS, Zhang YQ, Zhang LM, Liu C, Khurshid. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: ACM, 2018. 132–142. [doi: 10.1145/3238147.3238187]

[77] Zhou ZQ, Sun LQ. Metamorphic testing of driverless cars. Communications of The ACM, 2019, 62(3): 61–67. [doi: 10.1145/3241979]

[78] Lindvall M, Porter A, Magnusson G, Schulze C. Metamorphic model-based testing of autonomous systems. In: Proc. of the 2nd IEEE/ACM Int'l Workshop on Metamorphic Testing. Buenos Aires: IEEE, 2017. 35–41. [doi: 10.1109/MET.2017.6]

[79] Wang LY. Research of test case generation technology on metamorphosis test [MS. Thesis]. Changchun: Jinlin University, 2018 (in Chinese with English abstract).

[80] Akgün O, Gent IP, Jefferson C, Miguel I, Nightingale P. Metamorphic testing of constraint solvers. In: Proc. of the 24th Int'l Conf. on Principles and Practice of Constraint Programming. Lille: Springer, 2018. 727–736. [doi: 10.1007/978-3-319-98334-9_46]

[81] Tolksdorf S, Lehmann D, Pradel M. Interactive metamorphic testing of debuggers. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Beijing: ACM, 2019. 273–283. [doi: 10.1145/3293882.3330567]

[82] Olsen M, Raunak M. Increasing validity of simulation models through metamorphic testing. IEEE Trans. on Reliability, 2019, 68(1): 91–108. [doi: 10.1109/TR.2018.2850315]

[83] Dwarakanath A, Ahuja M, Sikand S, Rao RM, Bose RPJC, Dubash N, Podder S. Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In: Proc. of the 27th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Amsterdam: ACM, 2018. 118–128. [doi: 10.1145/3213846.3213858]

[84] Zhang ZY, Wang P, Guo HJ, Wang ZY, Zhou YQ, Huang ZQ. DeepBackground: Metamorphic testing for deep-learning-driven image recognition systems accompanied by background-relevance. Information and Software Technology, 2021, 140: 106701. [doi: 10.1016/j.infsof.2021.106701]

[85] Wildandyawan A, Nishi Y. Object-based metamorphic testing through image structuring. arXiv:2002.07046, 2020.

[86] Jiang MY, Chen TY, Zhou ZQ, Ding ZH. Input test suites for program repair: A novel construction method based on metamorphic relations. IEEE Trans. on Reliability, 2021, 70(1): 285–303. [doi: 10.1109/TR.2020.3003313]

[87] Moreira D, Furtado AP, Nogueira S. Testing acoustic scene classifiers using metamorphic relations. In: Proc. of the 2020 IEEE Int'l Conf. on Artificial Intelligence Testing. Oxford: IEEE, 2020. 47–54. [doi: 10.1109/AITEST49225.2020.00014]

[88] Li Z, Cui ZQ, Liu JB, Zheng LW, Liu XL. Testing neural network classifiers based on metamorphic relations. In: Proc. of the 6th Int'l Conf. on Dependable Systems and Their Applications (DSA). Harbin: IEEE, 2020. 389–394. [doi: 10.1109/DSA.2019.00060]

[89] Troup M, Yang A, Kamali AH, Giannoulatou E, Chen TY, Ho JWK. A cloud-based framework for applying metamorphic testing to a bioinformatics pipeline. In: Proc. of the 1st IEEE/ACM Int'l Workshop on Metamorphic Testing. Austin: IEEE, 2016. 33–36. [doi: 10.1109/MET.2016.014]

[90] de Andrade SA, Santos Í, Junior CB, Júnior M, de Souza SRS, Delamaro ME. On applying metamorphic testing: An empirical study on academic search engines. In: Proc. of the 4th IEEE/ACM Int'l Workshop on Metamorphic Testing. Montreal: IEEE, 2019. 9–16. [doi: 10.1109/MET.2019.00010]

[91] Ayerdi J, Segura S, Arrieta A, Sagardui G, Arratibel M. QoS-aware metamorphic testing: An elevation case study. In: Proc. of the 31st IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Coimbra: IEEE, 2020. 104–114. [doi: 10.1109/ISSRE5003.2020.00019]

[92] Rothermel J, Lindvall M, Porter A, Bjorgvinsson S. A metamorphic testing approach to NASA GMSEC's flexible publish and subscribe functionality. In: Proc. of the 3rd IEEE/ACM Int'l Workshop on Metamorphic Testing. Gothenburg: IEEE, 2018. 18–25. [doi: 10.1145/3193977.3193984]

[93] Hui ZW, Huang S. Experience Report: How do metamorphic relations perform in geographic information systems testing. In: Proc. of the 40th IEEE Annual Computer Software and Applications Conf. Atlanta: IEEE, 2016. 598–599. [doi: 10.1109/COMPSAC.2016.112]

[94] Brown J, Zhou ZQ, Chow YW. Metamorphic testing of navigation software: A pilot study with Google maps. In: Proc. of the 51st Annual Hawaii Int'l Conf. on System Sciences. Waikoloa Village: AIS, 2018. 5687–5696.

[95] Lin XY, Simon M, Niu N. Hierarchical metamorphic relations for testing scientific software. In: Proc. of the 2018 Int'l Workshop on Software Engineering for Science. Gothenburg: ACM, 2018. 1–8. [doi: 10.1145/3194747.3194750]

[96] Zhang M, Keung JW, Chen TY, Xiao Y. Validating class integration test order generation systems with metamorphic testing. Information and Software Technology, 2021, 132: 106507. [doi: 10.1016/j.infsof.2020.106507]

[97] Jarman DC, Zhou ZQ, Chen TY. Metamorphic testing for adobe data analytics software. In: Proc. of the 2nd IEEE/ACM Int'l Workshop on Metamorphic Testing. Buenos Aires: IEEE, 2017. 21–27. [doi: 10.1109/MET.2017.1]

[98] Cañizares PC, Núñez A, De Lara J. An expert system for checking the correctness of memory systems using simulation and metamorphic testing. Expert Systems with Applications, 2019, 132: 44–62. [doi: 10.1016/j.eswa.2019.04.070]

[99] Almendros-Jiménez JM, Becerra-Terón A, Merayo MG, Núñez M. Metamorphic testing of OpenStreetMap. Information and Software Technology, 2021, 138: 106631. [doi: 10.1016/j.infsof.2021.106631]

[100] Tu KY, Jiang MY, Ding ZH. A metamorphic testing approach for assessing question answering systems. Mathematics, 2021, 9(7): 726. [doi: 10.3390/math9070726]

[101] Tian T, Gong DW. Survey on mutation testing of concurrent programs. Acta Electronica Sinica, 2020, 48(11): 2267–2277 (in Chinese with English abstract). [doi: 10.3969/j.issn.0372-2112. 2020.11.025]

[102] Baidu Wenku. Advanced operating system—Su Shuguang. 2014 (in Chinese). https://wenku.baidu.com/view/b6347ebbad51f01dc281f1b0.html

附中文参考文献:

[3] 刘佳洛, 姚奕, 黄松, 陈强, 寇大磊, 张仲伟. 机器学习图像分类程序的蜕变测试框架. 计算机工程与应用, 2020, 56(17): 69–77. [doi: 10.3778/j.issn.1002-8331.1906-0242]

[11] 董国伟, 徐宝文, 陈林, 聂长海, 王璐璐. 蜕变测试技术综述. 计算机科学与探索, 2009, 3(2): 130–143. [doi: 10.3778/j.issn.1673-9418.2009.02.002]

[14] 惠战伟, 黄松, 李辉, 刘剑豪. 蜕变关系形式化描述与分解技术. 计算机工程与设计, 2016, 37(2): 405–412. [doi: 10.16208/j.issn1000-7024.2016.02.024]

[25] 张兴隆, 于磊, 侯雪梅, 侯韶凡. 面向对象程序蜕变关系构造方法. 计算机科学, 2017, 44(S2): 485–489, 515.

[26] 侯雪梅, 于磊, 张兴隆, 李志博. 面向对象软件测试的蜕变关系构造方法. 计算机应用, 2015, 35(10): 2990–2994.

[33] 穆翔宇, 范钰, 李苏吉, 张鹏, 刘磊. 一种基于梯度下降算法的蜕变关系生成方法. 吉林大学学报(理学版), 2020, 58(6): 1429–1435. [doi: 10.13413/j.cnki.jdxblxb.2019456]

[40] 董国伟, 聂长海, 徐宝文. 基于程序路径分析的有效蜕变测试. 计算机学报, 2009, 32(5): 1002–1013. [doi: 10.3724/SP.J.1016.2009.01002]

[41] 张兴隆, 于磊, 侯雪梅, 李志博, 李刚. 基于差异度分析的蜕变关系选取方法. 计算机工程与设计, 2017, 38(1): 103–109. [doi: 10.16208/j.issn1000-7024.2017.01.020]

[43] 张月. 数值程序蜕变关系构造及优化研究 [硕士学位论文]. 厦门: 华侨大学, 2019.

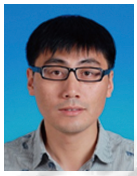
[44] 刘利娟. 蜕变关系有效性的研究 [硕士学位论文]. 长春: 吉林大学, 2018.

[49] 王琼, 贲可荣. 蜕变关系构造基本准则与策略研究. 计算机科学, 2012, 39(1): 115–119. [doi: 10.3969/j.issn.1002-137X.2012.01.026]

[51] 谢晓东, 彭声明, 刘艳, 汪康炜, 王田, 王成. 蜕变关系敏感度及其聚类分析. 电子学报, 2016, 44(5): 1196–1201. [doi: 10.3969/j.issn.0372-2112.2016.05.026]

[53] 吴鹏,施小纯,唐江峻,林惠民,陈宗岳. 关于蜕变测试和特殊用例测试的实例研究. 软件学报, 2005, 16(7): 1210-1220. http://www.jos.org.cn/1000-9825/16/1210.htm [doi: 10.1360/jos161210]
[67] 范超. 似然蜕变关系动态发现工具设计及其应用研究 [硕士学位论文]. 衡阳: 南华大学, 2018.
[68] 时小芳. 基于线性蜕变关系的蜕变测试工具设计及应用研究 [硕士学位论文]. 衡阳: 南华大学, 2018.
[74] 钟文康, 葛季栋, 陈翔, 李传艺, 唐泽, 骆斌. 面向神经机器翻译系统的多粒度蜕变测试. 软件学报, 2021, 32(4): 1051–1066. http://www.jos.org.cn/1000-9825/6221.htm [doi: 10.13328/j.cnki.jos.006221]
[79] 王丽瑶. 蜕变测试的测试用例生成技术的研究 [硕士学位论文]. 长春: 吉林大学, 2018.
[101] 田甜, 巩敦卫. 并发程序变异测试研究综述. 电子学报, 2020, 48(11): 2267–2277. [doi: 10.3969/j.issn.0372-2112. 2020.11.025]
[102] 百度文库. 高级操作系统—苏曙光. 2014. https://wenku.baidu.com/view/b6347ebbad51f01dc281f1b0.html

田甜(1987－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为程序分析与测试.



于旭(1981－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为物联网, 推荐系统, 迁移学习.

杨秀婷(1997－), 女, 硕士, 主要研究领域为并行程序软件测试.

巩敦卫(1970－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为智能优化, 基于搜索的软件工程.

王安轼(1998－), 男, 硕士生, 主要研究领域为并行程序软件测试.