

**北 京 科 技 大 学**

本科生毕业设计(论文)选题报告

面向Java程序的并发变异体生成

题　　目： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

工具研究与实现

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

计算机与通信工程学院

学　　院： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

计算机科学与技术

专　　业： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

顾友达

姓　　名： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

41455112

学　　号： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

指导教师签字： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

年 月 日

目 录

[1 课题背景及研究意义 1](#_Toc513639866)

[1.1 课题背景 1](#_Toc513639867)

[1.2 研究意义 1](#_Toc513639868)

[2 文献综述 3](#_Toc513639869)

[2.1 变异测试研究现状 3](#_Toc513639870)

[2.2 并发变异测试研究现状 5](#_Toc513639871)

[3 研究内容、预期目标及研究方法 7](#_Toc513639872)

[3.1 研究内容 7](#_Toc513639873)

[3.1.1 并发变异测试 7](#_Toc513639874)

[3.1.2 具体内容 7](#_Toc513639875)

[3.2 预期目标 8](#_Toc513639876)

[3.3 研究方法 8](#_Toc513639877)

[3.3.1 文献研究法 8](#_Toc513639878)

[3.3.2 实例研究法 8](#_Toc513639879)

[4 研究进度安排 9](#_Toc513639880)

[参考文献 11](#_Toc513639881)

1. 课题背景及研究意义
   1. 课题背景

随着社会不断发展进步，计算机应用软件在我们的日常生活中越来越普遍，极大地提升了我们的学习和工作效率。软件作为计算机的核心，在计算机中起着举足轻重的作用。软件在设计和开发各个阶段都需要人的参与，难免会出现一些错误，尤其是在大型甚至超大型系统中，软件故障不可避免。软件的质量问题不仅会给用户带来不好的体验，甚至会对生命财产安全带来威胁。例如，在1996年由于软件故障造成了Ariance 5运载火箭发射失败[1]。

随着多核计算日益普及，并发程序充分利用计算资源并提升软件的处理效率。并发程序中存在多个并发执行的线程，线程之间通常显式或隐式地共享一些存储空间。线程间执行顺序的不确定性使得并发程序的设计、实现、测试和修复更加困难。如何保证并发软件的质量成为一个受到广泛关注的开放问题。

软件测试是一种通用且行之有效的软件质量保障手段，通过运行测试用例，比较测试用例的输出与预期输出是否一致来检测软件中潜藏的故障。由于并发程序执行的不确定性，揭示并发故障不仅需要执行具有揭错能力的测试用例还需要线程之间的交错符合某种特定的模式。

并发变异测试[2]是一种针对并发程序面向故障的测试技术，首先根据被测并发程序特征选择并发变异算子，然后应用选择的并发变异算子作用到源程序产生并发变异体。在识别出等价变异体后，如果现有的测试用例不能“杀死”所有非等价变异体，还需设计额外的测试用例并添加到测试用例集中以提高测试用例集的充分性。并发变异测试不仅可以评估测试用例集的充分性，也可以评估并发测试技术的有效性。

已有的面向Java语言的自动化变异测试工具（例如MuJava[3]、MuClipse[4]和Javalanche[5]等）不支持并发变异体的自动生成。由于测试资源受到限制，自动、快速地生成并发变异体是一个亟待解决的问题。

* 1. 研究意义

本课题的主要研究内容：根据Bradbury等人[6]提出的Java并发变异算子，开发能够支持并发变异体自动生成的工具，然后通过实例研究的方式验证工具的正确性和有效性。本课题的研究意义如下：

1. **提高并发变异体生成效率**：由于测试资源有限，工具生成并发变异体的方式在提升并发变异体的生成效率的同时有效减少测试人员工作量。
2. **辅助评估并发程序测试用例集的充分性**：自动生成并发变异体然后在并发变异体上执行测试用例得到变异评分，可以辅助测试人员评估针对待测程序的测试用例集的充分性。
3. **辅助评估并发程序测试用例集的有效性**：通过向待测程序植入故障模拟程序的真实故障，可以辅助评估测试人员针对并发程序测试提出的新型测试技术的有效性。
4. 文献综述

本章主要从变异测试研究现状和并发变异测试研究现状两方面简要概述一些相关的背景知识和目前一些主要的研究成果。

* 1. 变异测试研究现状

软件测试是保障软件质量和可靠性的一种重要手段，可以帮助开发或测试人员发现软件中的许多隐藏故障，避免这些潜在的故障影响用户的正常使用。软件测试的质量很大程度上由测试充分性决定，测试充分性准则是判定测试用例集对于被测程序是否充分的准则。如果测试用例集不充分，就必须增加更多的测试用例。这些准则不仅可以定量地规定软件测试需求，指导测试数据的选择，而且可以度量测试数据集揭示软件特定故障的能力，对测试结果和软件可靠性评估具有重要影响[7]。

变异测试是一种面向故障的软件测试技术，通过对被测程序进行语法改变，生成与原有程序差异极小的变异体来充分模拟被测程序可能的故障[8]。其可行性基于Demillo等人提出的两个重要的假设：熟练程序员假设和耦合效应假设[9]。

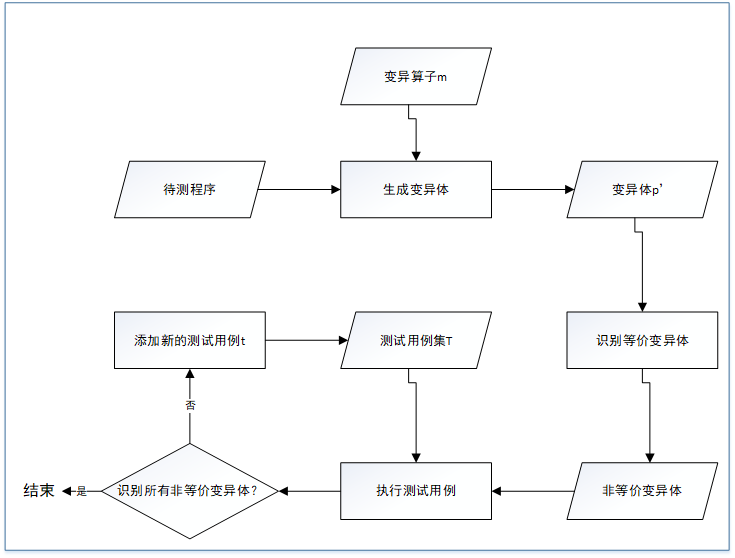


图1 变异测试基本流程

变异测试基本流程如图1所示，首先对据给定的被测程序p执行变异算子m然后生成大量的变异体。接下来从大量变异体中识别出等价变异体。然后在非等价变异体上执行测试用例集T，如果测试用例集T能够识别出所有非等价变异体，则变异测试结束，接受当前测试用例集。否则还需要添加测试用例t直至能够识别出所有的非等价变异体。

以下对于上述流程中的一些概念进行解释：

变异算子：在符合语法的前提下，变异算子定义了从被测程序到变异体之间的转化规则。

可杀死变异体：若存在测试用例t(t∈T)在变异体和原有程序p上的执行结果不一致，则称变异体相对于测试用例集T是可杀死变异体。

可存活变异体：若不存在测试用例t(t∈T)在变异体和原有程序p上的执行结果不一致，则称变异体相对于测试用例集T是可存活变异体。

等价变异体：如果变异体和被测程序p在语法上存在差异，但是语义上却完全一致，则称是p的等价变异体。

变异测试最终通过变异评分来评估测试用例的故障检查能力，变异评分计算方式如式2-1：

（式2-1）

表示测试用例集T杀死的变异体数量，|M|表示生成的所有的变异体数量，表示已识别的等价变异体数量，MS(M,T)表示变异评分，从表达式含义可以看出MS(M,T)取值为[0,1]，取值越接近1表明测试用例集T的故障检测能力越强。

变异测试技术不仅可以评估测试用例集的充分性，还能评估测试技术的有效性[10]。近年来，变异测试受到国内外学者地广泛关注并取得了大量的研究成果[11]，部分代表性的研究工作包括：

**在变异算子方面**：变异算子作为变异测试的基础，目前已有大量研究人员基于各种编程语言的特点设计了一系列的变异算子。根据作用语言的特点，变异算子可以分为两类：面向过程和面向对象的变异算子。

1. **面向过程编程语言的变异算子**：例如，Budd等人设计了面向Fortran语言的变异算子并开发相应支持工具PIMS[12]。Agrawal等人[13]根据C语言规范设计77个变异算子并划分为四类：变量变异、操作符变异、常量变异、语句变异；
2. **面向对象编程语言的变异算子：**相对于面向过程语言，面向对象语言引入了继承、封装、多态和异常处理等特性，使得传统变异算子不能适用面向对象的语言。Kim等人于1999年首先提出了20个专门针对Java语言的变异算子并引入了类变异的概念，随后又增加多态、重载、信息隐藏、异常处理类型的变异算子[14]。这些变异算子包含了面向过程语言的变异算子所没有的特性。

**在识别等价变异体方面**：变异得分建立在等价变异体识别完成之后，若不能检测出所有等价变异体，测试用例集充分性的评判就会受到影响。源代码在编译时借助优化规则可以生成语义等价代码，基于这一理论，Baldwin和Sayward提出编译器优化技术识别等价变异体的方法[15]。Adamopoulos等人提出一种基于遗传算法的协作演化法（即测试用例和变异体同时进行演化）来检测可能的等价变异体[16]。通过设置合理的适应值函数，确保当变异体是等价变异体时，该函数可以返回一个很小的适应值。基于该适应值函数，群体在演化过程中可以有效淘汰部分等价变异体，同时将那些难以检测的变异体和检测能力强的测试用例均保留下来。

* 1. 并发变异测试研究现状

并发变异测试就是将变异测试应用到并发程序中，通过对被测并发程序应用并发变异算子可生成大量的并发变异体，通过在并发变异体上执行测试用例并根据变异体的存活情况计算得到变异评分[17]。并发变异测试可以用来检测针对并发程序的测试用例集的充分性，同时也可以模拟被测程序的真实故障，帮助测试人员完善测试用例和改进测试技术[18]。本节从并发故障检测方面和并发变异算子研究方面简述目前一些相关的研究情况。

**并发故障检测方面**：与传统的顺序执行程序相比，并发程序的测试工作更具挑战性。由于线程之间执行的次序不确定，每一种线程执行序列对应一个交错模式，在某些交错模式下指令的执行顺序和开发人员的预期不同或者并发程序输出的结果与预期不同，这样的交错模式也称为交错性质。并发程序中常见的交错性质包括：（1）**原子违反：**某一线程中应具有原子性的一系列操作与来自其它线程的操作出现交错；（2）**死锁：**同步机制无限期地阻止一些线程继续执行；（3）**数据竞争：**当来自不同线程的两个操作（至少有一个是写操作）并发地访问没有同步机制保护的数据项。

近年来，围绕原子违反、死锁、数据竞争三类常见并发故障，研究者提出了一系列的并发程序测试技术。部分代表性研究工作包括：

1. **原子违反检测**：Lu等人提出了基于原子块划分的技术[19]，首先将程序划分成独立的原子块(atomic block)，然后分别检测破坏这些原子块缓存的行为。在Lu等人工作基础上，Lucia等人提出了ColorSafe技术[20]，将相关数据分组到相同的颜色（color）中，利用元数据来维护颜色信息和签名，以有效地保持最近的颜色访问历史，然后动态地监控颜色空间（color space）中的访问交错情况来检测原子违反故障。
2. **死锁检测**：Li等人[21]提出了动态内存监控技术，利用超时机制和动态记录资源使用情况检测是否存在死锁。Havelund[22]提出了Goodlock技术，通过动态地将每个程序执行的锁定模式记录为一棵锁树（lock tree），然后对线程的锁树进行比较从而发现可能导致死锁的循环依赖关系。Joshi等人根据Goodlock提出了DeadlockFuzzer技术[23]，DeadlockFuzzer利用从Goodlock收集的信息指导线程的调度，从而提高选择的线程交错模式检测出死锁的可能性。
3. **数据竞争检测**：Lipton[24]提出了锁集分析（Lockset analysis）技术，在访问共享数据时，通过动态监控所持有的公共锁集合的方法来识别没有使用任何锁就直接访问相同共享数据的执行流，从而检测出数据竞争故障。Lamport[25]提出了基于线程访问次序关系的检测技术happens-before，通过跟踪输入数据执行情况来检测潜在的数据竞争，相比于锁集分析happens-before技术可以运用于那些没有使用锁同步机制数据竞争检测。

**并发变异算子研究方面**：Richard Carver[26]最早将变异测试引入并发程序测试中，但是并未针对并发程序的故障专门设计变异算子。并发变异算子作为并发变异测试的基础，它定义了从并发程序到并发变异体的变换规则，在并发变异测试中起着举足轻重的作用。目前并发变异测试研究的许多工作都在并发变异算子研究方面[27]，并已取得了大量的成果。Delamaro通过分析Java语言特性和并发机制，提出了15种并发变异算子，并根据变异对象的不同将它们分成针对监听对象，针对等待集和针对方法调用三大类。Bradbury等人在Delamaro的工作基础上依据Java(J2SE 5.0)并发机制特性，设计出了包括修改并发方法参数，修改并发方法调用，修改关键字，交换并发对象，修改临界区等五类25个并发变异算子[6]。Wu等人则认为已有的并发变异算子存在一定的局限性，不能生成一些细微的并发故障，甚至一些变异算子可能无法产生任何变异体，为此，他们通过组合已有的一阶变异算子给出了6种二阶变异算子，并验证了其二阶变异算子确实能够生成一些一阶变异算子不能生成的并发故障。Markus等人设计出了针对C和C++的并发变异算子并根据这些并发变异算子开发出了并发变异测试工具CCmutator[28]。

1. 研究内容、预期目标及研究方法

本章主要介绍面向Java应用程序的并发变异测试支持工具的具体研究内容、预期目标及研究方法。

* 1. 研究内容
     1. 并发变异测试

1. **选取并发变异算子**：作为并发变异测试的基础，并发变异算子决定了具体的变异规则，对后续的测试用例评估有着至关重要的作用。本课题将深入分析Bradbury等人提出的Java(J2SE 5.0)并发变异算子并结合真实情况下Java程序员在编程中经常犯的一些并发错误，从中选取一部分最能代表编程人员易犯的并发错误的典型的变异算子。
2. **并发变异体生成工具设计与实现**：MuJava作为一个优秀的变异分析工具，已经提供了方法级别和类级别的变异分析支持，包括变异体生成、测试用例执行、变异评分等核心功能，同时也提供了良好的用户操作界面。但是目前并没有实现对并发变异测试的支持，所以本课题将在其基础上进行扩展，设计并开发出一个能够支持并发变异测试的自动化工具CT4Droid。
3. **测试验证**：为了确保工具生成并发变异体的正确性，利用实际Java并发应用程序对工具进行测试，验证自动生成的并发变异体是否正确。
   * 1. 具体内容
4. **分析选取并发变异算子**：相比于顺序程序，并发程序由于线程执行交错情况的不确定性会造成更多的潜在故障，例如死锁，数据竞争和原子违反等。为了模拟常见的真实故障，本课题将详细分析Bradbury等人提出的并发变异算子并结合真实场景选取部分比较有代表性的并发变异算子进行实现并利用实例进行验证。
5. **设计并实现并发变异测试工具CT4Droid**：在MuJava的基础上进行扩展，设计并开发出一个支持并发变异测试的工具CT4Droid。该工具能够实现并发变异体的自动生成，被测程序与变异体的对比分析，测试用例的执行，变异评分的计算等核心功能，同时也会提供良好的可操作用户界面。
6. **实例验证**：使用并发测试程序对CT4Droid工具进行实例验证，利用CT4Droid对被测程序进行变异生成并发变异体，然后对自动生成的并发变异体的正确性进行分析验证，确保工具的准确性和实用性。
   1. 预期目标

在MuJava的基础上进行扩展开发，设计并实现一个面向Java应用程序的并发变异体自动生成工具CT4Droid，支持根据选择的并发变异算子自动生成相应的并发变异体。

* 1. 研究方法
     1. 文献研究法

收集并阅读软件测试、变异测试和并发变异测试领域已有的相关研究文献资料，学习掌握变异测试的基本原理和实际应用，了解目前变异测试领域已有的研究成果和研究方向。

* + 1. 实例研究法

以变异测试工具MuJava为样例进行研究，深入阅读分析MuJava源码并掌握其变异体生成的原理和具体实现细节。在此基础上对MuJava进行扩展开发，设计并实现一个能够支持并发变异测试的自动化工具CT4Droid。利用CT4Droid对大量的并发程序进行变异生成并发变异体，然后对自动生成的并发变异体的正确性进行验证，确保工具的准确性与可用性。

1. 研究进度安排

第1-4周：学习并掌握变异测试基本原理、MuJava基本原理及实现、Java程序的并发机制、语法树生成等技术，完成开题报告。

第5-9周：根据Java并发变异算子，设计并实现一个面向Java程序的并发变异体自动生成的工具。

第10周：完成中期报告。

第11-12周：测试并完善并发变异体生成工具，完成翻译相关国外文献任务。

第13-15周：撰写论文，准备答辩。

学生本人签字：

年 月 日

参考文献

1. Weyuker E J. Testing component-based software: a cautionary tale [J]. IEEE Software, 1998, 15(5): 54-59.
2. Reales P, Polo M, Fernández-Alemán J L, Toval A，Piattini M. Mutation Testing [J]. IEEE Software, 2014, 31(3): 30-35．
3. Ma Y S, Offutt A J, Kwon Y R. MuJava: an automated class mutation system [J]. Software Testing, Verification and Reliability, 2005, 15(2): 97-133.
4. Smith B, Williams L. On guiding the augmentation of an automated test suite via mutation analysis [J]. Empirical Software Engineering, 2009, 14(3): 341-369.
5. Schuler D, Zeller A. Javalanche: efficient mutation testing for Java [C] //Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE 2009). ACM Press, 2009: 297-298.
6. Bradbury J S, Cordy J R, Dingel J. Mutation Operators for Concurrent Java (J2SE 5.0) [C] //Proceedings of the 2nd Workshop on Mutation Analysis, Co-located with the IEEE International Symposium on Software Reliability Engineering (ISSRE 2006). IEEE Computer Society, 2006: 83-92.
7. 单锦辉，姜瑛，孙萍．软件测试研究进展 [J]．北京大学学报（自然科学版），2005（01）：134-145．
8. Reales P, Polo M. Parallel Mutation Testing [J]. Software Testing Verification and Reliability, 2013, 23(4): 315-350.
9. Offutt A. Investigations of the software testing coupling effect [J]. ACM Transactions on Software Engineering and Methodology, 1992, 1(1): 5-20.
10. Bluemke I, Kulesza K. Reduction in Mutation Testing of Java Classes [C] //Proceedings of the 9th International Conference on Software Engineering and Applications (ICSOFT-EA 2014). IEEE Computer Society, 2014: 297-304.
11. 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.
12. Budd T A, Lipton R J, Demillo R, Sayward F G. The design of a prototype mutation system for program testing [C] //Proceedings of the AFIPS National Computer Conference (AFIPS 1978). ACM Press, 1978: 623-627.
13. Agrawal H, Demillo R A, Hathaway B. Design Of Mutant Operators For The C Programming Language [J]. The Fall of, 2001, 25(1): 14-27.
14. Kim S, Clark J A, McDermid J A. Class mutation: Mutation testing for object-oriented programs [C] //Proceedings of the Forum for Medical Ethics Society (FMES 2000), Springer, 2000: 9-24.
15. Baldwin D, Sayward F G. Heuristics for determining equivalence of program mutations [R]. Yale University, 1979.
16. Adamopoulos K, Harman M, Hierons R M. How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution [C] //Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Springer, 2004: 1338-1349.
17. Carver R. Selective Mutation Testing for Concurrent Code [C] //Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2013). ACM Press, 2013: 224-234.
18. Cao L, Zheng W, Hu D. Concurrent Program Semantic Mutation Testing Based on Abstract Memory Model [C] //Proceedings of IEEE International Conference on Information and Automation (ICIA 2015). IEEE Computer Society, 2015: 1200-1205.
19. Lu S, Tucek J, Qin F, Zhou Y. AVIO: Detecting Atomicity Violations via Access-Interleaving Invariants [J]. IEEE Micro, 2007, 27(1): 26-35.
20. Lucia B, Ceze L, Strauss K. ColorSafe: architectural support for debugging and dynamically avoiding multi-variable atomicity violations [J]. ACM Sigarch Computer Architecture News, 2010, 38(3): 222-233.
21. Li T, Ellis C S, Lebeck A R, Sorin D J. Pulse: a dynamic deadlock detection mechanism using speculative execution [C] //Proceedings of USENIX Annual Technical Conference (USENIX ATC 2005). USENIX Association, 2005: 31-44.
22. Havelund K. Using Runtime Analysis to Guide Model Checking of Java Programs [C] //Proceedings of International SPIN Workshop on Model Checking of Software. Springer (SPIN 2000), 2000: 245-264.
23. Joshi P, Park C S, Sen K, Naik M. A randomized dynamic program analysis technique for detecting real deadlocks [C] //Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2009), 2009: 110-120.
24. Lipton R J. Reduction: A method of proving properties of parallel programs [J] Communications of the ACM, 1975,18(12): 717–721.
25. Lamport L. Time, clocks, and the ordering of events in a distributed system [J]. Communications of the ACM, 1978, 21(7): 558-565.
26. Carver R. Mutation-based testing of concurrent programs [C] //Proceedings of International Test Conference (ITC 1993). IEEE Computer Society, 1993: 845-853.
27. 茆亮亮．变异测试技术应用研究 [D]．中国科学技术大学，2010．
28. Kusano M, Wang C. CCmutator: A mutation generator for concurrency constructs in multithreaded C/C++ applications [C] //Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013). IEEE Computer Society, 2013: 722-725.

指导教师意见

指导教师签字：

年 月 日