CMuJava: 一个面向 Java 程序并发变异测试支持系统*

孙昌爱, 耿 宁, 代贺鹏, 顾友达

(北京科技大学 计算机与通信工程学院,北京 100083)

通讯作者: 孙昌爱, E-mail: casun@ustb.edu.cn

摘 要: 并发系统由多个并发执行的流程组成,并且流程之间通常显示或隐式地共享一些存储空间.流程之间执行次序的不确定性给并发系统的测试带来了严峻的挑战.并发变异测试是一种基于并发故障的软件测试技术,依据并发变异算子改变待测程序的语法结构,生成并发变异体,通过执行并发变异体来评估测试用例集的充分性和测试技术的有效性.然而,面向 Java 程序的开源变异测试工具不支持生成并发变异体,人工生成并发变异体需要消耗大量的测试资源.为了提高并发变异测试的效率,本文依据 Java 并发变异算子设计并实现了一个并发变异体自动生成工具 CMuJava,采用经验研究评估了 CMuJava 生成并发变异体的正确性、完备性及效率.经验研究的结果表明,该工具生成并发变异体的正确性和完备性均达到了 100%,并且与人工生成并发变异体相比,生成并发变异体的效率最高提高了约 8000 倍.

关键词: 并发系统;并发变异测试; 并发变异算子; 并发变异体; 测试工具

Abstract: Concurrent systems are composed of multiple concurrent execution flows, which usually share storage space explicitly or implicitly. Uncertainty of execution order among execution flows makes testing concurrent systems a challenging task. Concurrent mutation testing is a software testing technology based on concurrent faults, which changes the syntax of the program under test according to concurrent mutation operators, generates concurrent mutants to evaluate the adequacy of test case set and the effectiveness of test technology. However, current open source Java-oriented mutation testing tools do not support concurrent mutants generation, and manual generation of concurrent mutants requires a lot of test resources. In order to improve the efficiency of concurrent mutation testing, we designs and implements a concurrent mutation automatic generation tool CMuJava in this paper, which is based on Java concurrent mutation operators, and evaluates the correctness, completeness and efficiency of CMuJava generating concurrent mutants by empirical research. The results of the empirical study show that the correctness and completeness of the tool for generating concurrent mutants are 100%, and the efficiency of generating concurrent mutants is about 8000 times higher than that of generating concurrent mutants manually.

Key words: concurrent system; concurrent mutation testing; concurrent mutation operator; concurrent mutant; testing tools

随着多核计算的日益普及,并发程序已经成为人们广泛关注与重视的研究领域.并发程序中存在多个并发执行的流程,流程之间通常显示或者隐式地共享一些存储空间,并且它们的执行次序不确定[1].并发流程之间不确定的相互作用与影响的情形称为执行交错.执行交错的存在使得在完全相同的环境下运行两次同样程序的输出结果可能不同,并且程序故障难以重现,给并发程序的测试带来了严峻的挑战.如何有效地检测并发程序中潜藏的故障,提高并发程序的可靠性成为一个亟待解决的重要问题.

变异测试[2]是一种基于故障的软件测试技术,可以用来评估测试用例集的充分性与测试技术的有效性.测试人员通过分析原始待测程序并设计出一系列变异算子[3],然后对待测程序应用变异算子来生成大量的变异体[3]。在识别出等价变异体[3]后,若已有测试用例不能杀死所有非等价变异体[3],则需要设计新的测试用例来提高测试充分性.

Carver 将变异测试理论和技术应用到并发程序测试中,提出了并发变异测试[4].在并发变异体生成方面,面向 Java 语言的开源变异测试工具(例如 MuJava[5]、Javalance[6]、Jumble[7]和 Major[8]等)不支持生成并发变异体,已有的并发变异体生成工具 Paraμ[9]目前仅实现了 3 个并发变异算子,而 Comutation[10]工具不支持开源使用.研究者在测试过程中通常依据并发变异算子人工生成并发变异体,该方式需要占用大量的测试资

* 基金项目:

Foundation item:

源.因此,需要一个高效的并发变异测试支持系统来提高并发变异测试的自动化程度.开发面向并发程序的变异测试系统面临如下挑战:(1)提高并发变异体生成效率.一些并发变异算子定义的规则具有多样性,即针对同一并发机制可以产生大量变异体,并发变异系统需要多次分析源程序的结构,运用相应的规则改变源程序并生成并发变异体;(2)提高并发系统的可扩展性.随着并发理论与相关技术的不断发展,并发程序越来越复杂,可能出现新的并发变异算子来模拟并发故障.因此,实用的并发变异系统需要具备高效地声生成并发变异体以及可扩展的特点.

MuJava 是一个常用的面向 Java 程序的变异测试工具[11,12,13],其自身具有很好的可扩展性,例如 Sun 等人将非均匀分布的变异分析技术[14]应用到 MuJava,扩展出支持非均匀分布的变异生成系统 MuJavaX[15];Kim 等人通过扩展 MuJava 实现了所提的弱变异与强变异结合的方法[16].由于 MuJava 不能基于并发故障生成并发变异体,本文通过扩展 MuJava 设计并实现了一个自动生成并发变异体的工具 CMuJava,提高了面向 Java 程序的并发变异测试技术的自动化程度.本文的主要贡献有:

(1) 基于 Bradbury 等人提出的并发变异算子扩展了变异测试工具 MuJava,实现了一个在能够生成传统变异体的基础上自动生成并发变异体的工具 CMuJava.
(2) 采用反射技术和 MSG 结合的方法提高了并发变异体的生成效率。
(3) 应用 Visitor 设计模式提高了工具的可扩展性.
(4) 以经验研究的方式评估了 CMuJava 生成并发变异体的正确性、完备性和效率.

本文第 1 节介绍了并发变异测试相关的背景知识.第 2 节讨论工具的设计与实现.第 3 节以经验研究的方式评估工具生成并发变异体的正确性、完备性和效率.第 4 节对实验结果进行了分析.第 5 节介绍与本文相关的研究工作.最后,第 6 节对本文的工作做出了总结并提出将来需要完善的方面.

1 并发变异测试

在传统变异测试中,首先,应用变异算子改变原始程序 P 的语法结构,生成与原始程序有语法差异的变异体集合$\{P_1', P_2', \dots, P_n'\}$来模拟待测程序可能存在的故障.然后,在原始程序 P 和每个变异体$P_i'(i = 1,2, \dots, n)$上执行所有的测试用例,如果存在测试用例在 P 和P_i'上的执行结果不同,则称变异体P_i'为可杀死变异体,如果所有的测试用例在 P 和P_i'上执行的结果完全一致,则称P_i'为 P 的等价变异体.最后,通过式(1)计算变异得分,评估测试用例集的充分性和测试技术的有效性.

$$MS(M,T) = \frac{killed(M,T)}{\{|M| - eqv(M)\}} \quad (1),$$

其中,$MS(M,T)$表示变异得分,$killed(M,T)$表示可杀死变异体数量,$|M|$表示生成的所有变异体数量,$eqv(M)$表示已识别的等价变异体数量.

与顺序程序相比,并发程序中多个执行流之间不确定的执行交错为测试并发程序带来了严峻的挑战.很多研究者从不同的角度出发,针对并发程序提出了一系列故障检测技术,例如:锁集分析[17]、happens-before[18]和并发变异测试技术[4].其中,并发变异测试将传统变异测试技术应用到并发程序中,模拟真实的并发故障,可以有效地提高故障检测效率.

传统变异测试定义为 5 元组$E = (P,S,D,L,A)$[19],其中,P 是原始程序,S 是规格说明,D 是测试用例集,$L = (l_1, l_2, \dots, l_n), l_i(i = 1,2, \dots, n)$表示 P 中语句的位置,$A = (A_1, A_2, \dots, A_n), A_i(i = 1,2, \dots, n)$是位置$l_i$处的变异算子集合,$|A_i|$是变异算子的数量.

与传统变异测试不同,并发变异测试是对程序中特有的并发机制应用变异算子.在传统变异测试模型的基础上,我们用 C 表示 P 中的并发机制所在的位置,将并发变异测试定义为 6 元组$E = (P,S,D,L,C,A)$,其中,P 是原始并发程序,S 是规格说明,D 是测试用例集,$L = (l_1, l_2, \dots, l_n), l_i(i = 1,2, \dots, n)$表示 P 中语句的位置,$C = (c_1, c_2, \dots, c_n), c_i(i = 1,2, \dots, n)$表示 P 中并发机制的位置,$A = (A_1, A_2, \dots, A_n), A_i(i = 1,2, \dots, n)$是位置$c_i$

的变异算子集合,$|A_i|$是变异算子的数量.

图 1 描述了并发变异测试的流程,主要步骤如下:

(1) 分析原始并发程序中所有的并发机制.

(2) 根据原始并发程序所运用的并发机制选择合适的并发变异算子,然后在待测并发程序上应用并发变异算子生成相应的并发变异体.

(3) 从生成的大量并发变异体中识别出等价并发变异体并删除.

(4) 在非等价并发变异体上执行测试用例.

(5) 根据并发变异体被识别的情况计算变异得分.如果变异得分满足要求则变异测试结束,否则进行步骤(6).

(6) 设计新的测试用例并添加到当前测试用例集,回到步骤(4).

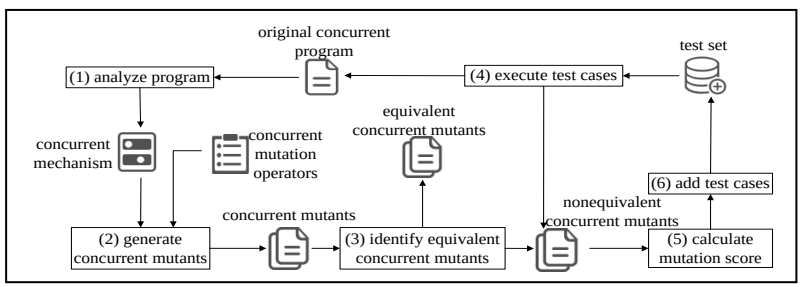

Fig.1 Concurrent Mutation Testing process

图 1 并发变异测试流程

Bradbury 等人[20]依据 Java(J2SE 5.0)并发机制特性及 Java 并发程序的故障类型[21]提出了面向 Java 程序的 25 种并发变异算子.这 25 种变异算子反应了真实的并发故障,具有代表性和良好的覆盖性.并发变异算子作为并发变异测试的基础,定义了改变待测并发程序语法结构的规则,表 1 列出了 25 种并发变异算子的信息,详细的并发变异算子信息和并发变异体示例可以在文献[20]和技术报告[22]中找到.

并发变异测试执行过程主要包括并发变异体生成和并发变异体执行两个阶段.针对复杂的并发程序,应用并发变异算子生成的变异体数量很大,人工生成需要消耗大量的资源,同时执行大量的并发变异体的速度缓慢.为了提高并发变异测试执行效率,需要设计自动化的并发变异分析系统.Gligoric 等人提出了一个变异测试优化工具 MutMut[23],他们应用了 4 种技术优化了并发变异体的执行,据报道该工具对变异测试的时间降幅达 77%,然而由于缺少并发变异体生成工具,实验中应用到的并发变异体均由人工生成.Madiraju 等人[9]提出了一种部分变异的方法,选择程序性中的复杂部分或方法进行变异,实现的工具 Paraμ 应用该方法减少了并发变异体的数量,但是工具只实现了 3 个并发变异算子,不利于广泛使用.Gligoric 等人[10]提出的 Comutation 应用了选择变异的方法提高并发变异测试效率,即应用并发变异算子的子集生成并发变异体,从而减少了并发变异体的数量.Paraμ 和 Comutation 均通过减少并发变异体数量提高了变异测试效率,但是对于并发变异体的生成,仍然通过修改原始代码的副本的方法单独地生成每个变异体,这种方法的低效率导致变异测试整个过程缓慢.

Table 1 concurrent mutation operators for Java[20]
表 1 Java 并发变异算子[20]

变异算子类别	并发变异算子
修改并发方法参数	MXT—修改方法参数 MSP—修改同步代码块参数 ESP—交换同步块参数 MSF—修改信号量公平性 MXC—修改权限许可和线程数量 MBR—修改同步屏障参数
修改并发方法调用	RTXC—删除线程方法调用 RCXC—删除并发机制方法调用 RNA—使用 notify()替换 notifyAll() RJS—使用 join()替换 sleep() ELPA—交换权限/锁获取方法 EAN—将原子调用替换为非原子调用
修改关键字	ASTK—为方法添加 static 关键字 RSTK—删除方法 static 关键字 ASK—为方法添加 synchronized 关键字 RSK—删除方法 synchronized 关键字 RSB—删除 synchronized 代码块 RVK—删除 volatile 关键字 RFU—删除 unlock 方法所在的 finally 代码块
交换并发对象	RXO—替换并发机制 EELO—交换锁对象
修改临界区	SHCR—移动临界区 SKCR—收缩临界区 EXCR—扩大临界区 SPCR—拆分临界区

2 CMuJava 的设计与实现

基于 Bradbury 等人提出的 25 种并发变异算子,我们开发了一个面向 Java 程序的并发变异体自动生成工具 CMuJava.工具采用反射技术[24,25]分析原始程序中的并发机制,并采用 MSG(Mutant Schema Generation)[26]方法提高并发变异体的生成效率.接下来详细讨论 CMuJava 设计与实现的关键问题.

2.1 CMuJava的基本原理

图 2 描述了 CMuJava 的基本原理.首先,工具读取原始并发程序及其依赖程序,按照 Java 程序的语法结构拆分并进行语法分析,得到程序中的并发机制;然后,对程序中的并发机制应用并发变异算子,依据变异算子定义的规则对程序进行修改生成并发变异体;最后,对原始程序和生成的并发变异体执行测试人员编写的 JUnit 测试脚本,自动获取测试结果,统计最终的变异得分以及每一个变异体的“杀死”信息.

生成变异体的关键步骤是按照变异算子所定义的规则对待测程序进行修改,所以首先需要对待测程序进行解析并获取待测程序的语法信息.传统的面向过程程序的变异分析系统中用一个普通的语法分析器解析原始程序,语法分析器生成抽象语法树来表示源代码,通过修改抽象语法树生成变异体.但是,传统的抽象语法树不能直接访问程序的面向对象特征(例如继承关系以及一些属性和方法的定义)[27],Chevalley 提出用反射方法解决上述传统语法分析技术不适用于面向对象程序变异分析的问题[27].反射技术允许程序访问其内部结构和行为并操纵该程序,从而根据另一程序提供的规则修改其行为[5].反射技术被应用到变异分析中的原因有[5]:(1)提供了表示某个 class 对象逻辑结构的引用,允许程序员提取该 class 面向对象的相关信息,这利

于对原始程序进行解析;(2)提供了一个 API,可以在执行过程中更改程序的行为,有助于变异体生成;(3)允许实例化对象并动态调用对象的方法.

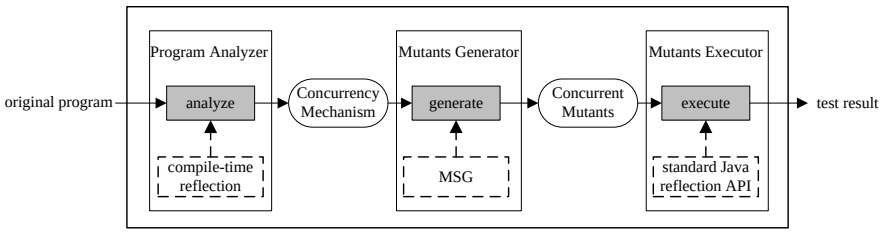

Fig.2 Basic principles of CMuJava
图 2 CMuJava 基本原理

在并发变异测试中,为了应用并发变异算子,系统需要解析出原始程序中的并发机制,这涉及到检测程序中面向对象的特征.因此,我们将反射技术应用到并发变异分析系统 CMuJava 中.Java 语言使用专用 API 提供内置的反射功能,允许 Java 程序执行诸如请求给定对象的类,查找该类中的方法以及调用这些方法等功能.但是,Java 语言提不支持完整的反射功能,只可以修改数据结构但不支持更改程序的行为[5].研究者已经提出了几种反射系统[28,29,30,31]来补充 Java 反射 API 的缺陷.其中,OpenJava[28]是一个编译时反射系统,其主要思想是用元程序处理表示程序逻辑实体的类元对象.OpenJava 在编译时使用元信息从而提供了比运行时反射系统更好的性能.我们在 CMuJava 中使用了 OpenJava 对原始并发程序进行语法解析,解析完成后会将待测程序的所有信息存储到一个元对象中.每一个程序在解析之后都会得到一个对应的元对象.元对象的设计与传统的抽象语法树设计思想相似,都是通过一个树状的结构来表达程序的语法信息,不同之处在于元对象可以表示抽象语法树无法表示的程序逻辑结构信息.因此,可以有效解决解析并发机制的问题.

在生成并发变异体阶段,传统的方法只能依据并发变异算子的规则修改原始代码的副本单独地生成每个变异体[5],生成速度缓慢.另外,由于需要存储和编译大量变异体,对空间的需求也比较大.我们采用 MSG 技术降低生成并发变异体的空间开销,进而提高并发变异测试的效率.图 3 描述了生成和执行并发变异体的过程.

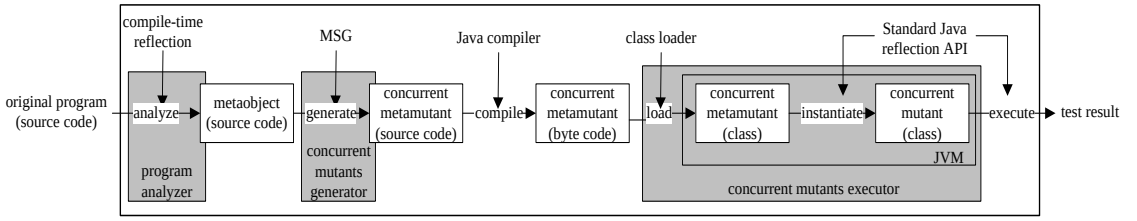

Fig.3 Mutation process for concurrent mutants
图 3 并发变异体的变异分析过程

MSG 技术可以将一个程序的所有变异体都编码到一个特殊的参数化程序中,这个特殊的参数化程序叫作元变异体.元变异体包含了待测程序所有变异体的信息和功能,在减少了空间开销的同时保证了运行过程中编译速度不变.下面我们以一个并发变异体的例子简单介绍元变异体相关内容.

例子: MSP(Modify Synchronized Parameter)变异算子定义的规则是更改 synchronized 代码块的同步参数,将其由“this”关键字替换为其他对象,或者由其他对象修改为“this”关键字.从 MSP 的变化规则可以

知道,每次被更改的地方都只有 synchronized 的同步参数,我们将源程序的 synchronized 代码块抽象为 Synchronized (getParameter(orignalArg, objList)){…},这样在程序运行的时候就可以通过 getParameter 方法动态的获取同步参数并生成相应的变异体.getParameter 方法有两个参数,其中“originalArg”表示的是原来的同步参数,“objList”表示的是可以用于替换原来的同步参数的对象列表.在对一个并发程序应用 MSP 并发变异算子时,CMuJava 在确定当前程序符合 MSP 并发变异规则之后,会自动获取程序的所有成员变量.然后将 synchronized 代码块的原始参数和成员变量列表传递给此方法,就可以生成待测程序的所有 MSP 变异体.将程序中的语句改变成这种形式后的参数化程序即为元变异体.

2.2 CMuJava的设计与实现

目前存在多个自动化的变异测试工具,具有代表性的有 MuJava、Javalance、Jumble、Major 等.其中,MuJava 是一个支持传统变异体生成的开源变异分析系统,具有良好的代表性和可扩展性.我们选择集成和扩展 MuJava 的功能设计与实现 CMuJava,使其在能够生成传统变异体的基础上支持并发变异体的生成.本节首先描述了 CMuJava 的系统架构,然后讨论了系统实现的关键问题.

1. CMuJava 架构

图 4 展示了 CMuJava 的架构.系统主要包括三个组件:

(1) 变异体生成组件可以根据传统的变异体算子和并发变异算子分别生成传统的变异体和并发变异体.该模块采用访问者设计模式实现,支持根据不同的变异算子(传统变异算子和并发变异算子)生成变异体,并且具有易于拓展的特点,即在不改动整体架构的情况下,添加新的变异算子.测试人员通过该组件的图形化界面选择待测程序和变异算子.

(2) 变异体查看组件一方面展示变异体生成组件产生的变异体信息,包括:生成的变异体数量和变异体类型.该组件也支持向测试人员展示选中的变异体与源程序的差异,帮助测试人员为难“杀死”的变异体设计测试用例和识别等价变异体.

(3) 变异体执行组件负责执行测试用例并输出测试用例集的变异得分.CMuJava 系统中,每一个变异体通过 Javac 编译并形成相应的类文件,然后该工具通过 Java 虚拟机(JVM)执行类加载器加载的类文件中的字节码进而执行变异体.因此,一个测试用例是由一系列调用待测程序方法的指令组成.

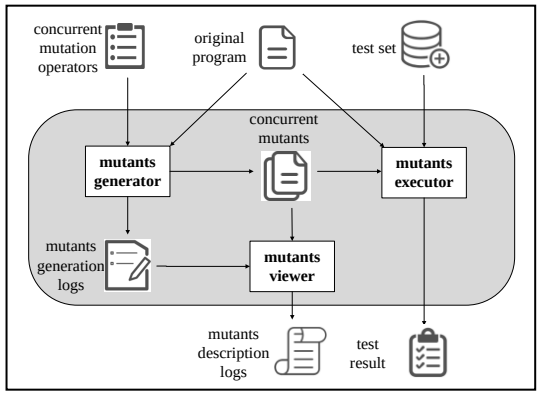

Fig 4 The architecture of MuJava

图 4 CMujava 架构

2. 增加系统的可扩展性

并发变异算子可以模拟程序中真实的并发故障,随着并发程序的发展,可能会发现新的并发故障,因此会需要新的并发变异算子.目前,CMuJava 工具实现了 25 种并发变异算子,为了将来可以应用新的并发变异算

子,我们使用了访问者(Visitor)设计模式[32]提高工具的可扩展性.

CMuJava 在完成一个 Java 程序解析得到一个元对象之后,需要在此对象的各个节点上运用一系列并发变异算子进行可变异点查找,然后对可变异点进行语法更改生成并发变异体.如果将所有的可变异点判断逻辑和变异体输出逻辑都放到元对象的每个节点内部,每增加一个变异算子都需要对元对象内部的一个或多个节点的操作方法进行修改以完成此变异算子的功能,这样会导致相关节点的代码变得越来越复杂,在降低了代码的可读性的同时也会增加后期的维护成本,同时不易于扩展.

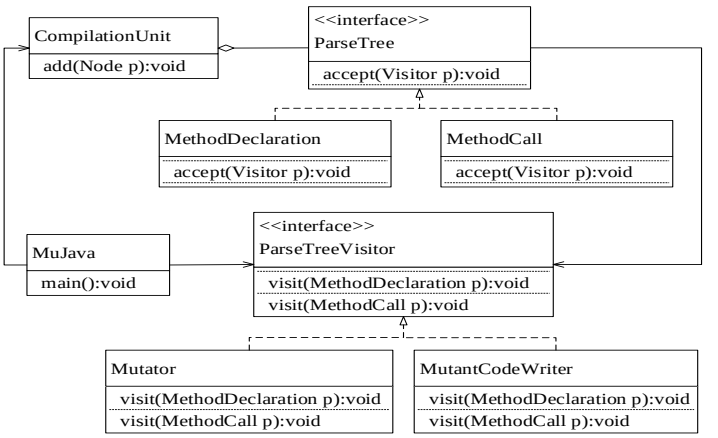

Fig.5 The core class diagram of CMuJava

图 5 CMuJava 核心类图

访问者设计模式适用于被访问对象结构相对稳定,且访问者操作不确定的情况.CMuJava 系统所涉及的元对象的数据结构非常稳定,不存在需要新加入节点或者对已有节点进行修改的情况,但是存在到对元对象的大量不确定操作,涉及到众多并发变异算子.为了提高工具的可扩展性即便于添加新的变异算子,我们在 CMuJava 实现过程中采用了访问者设计模式.

图 5 描述了 CMuJava 的核心类结构.其中,CompilationUnit 充当结构对象的角色,它负责存储所有的节点对象,并提供了相应的方法来遍历节点对象.访问者类 Mutator 和 MutatorWriter 均实现了 ParseTreeVisitor 类声明的对所有的节点的 visit 方法.当有新的并发变异算子出现需要添加到系统中时,只需要编写新的访问者类继承 Mutator 和 MutatorWriter 类,然后重写自己需要访问的节点的 visit 方法即可,极大地提高了工具的可扩展性.

2.3 工具演示

本文基于 Java 开发了工具原型,图 6 展示了工具的界面.界面左侧列出了所有原始待测程序,首先,测试人员在“File”栏中选择需要生成变异体的原始程序(图中以实验对象 ThreadID 程序为例);然后在“Java Mutation Operator”栏中选择原始程序适用的变异算子,其中最左侧为 19 种方法级别变异算子,中间为 28 种类级别变异算子,最右侧为本文实现的 25 种并发级别变异算子;最后,测试人员点击最下方黄色的“Generate”按钮,等待系统完成变异体生成工作,系统会依次读取选择的待测程序,为每个程序生成相应的元对象,然后在每个元对象上依次应用选中的变异算子生成相应的变异体并输出到指定目录.

变异体生成完成后,CMuJava 提供了查看变异体的功能,例如,点击“Concurrent Mutants Viewer”可以查看生成的所有的并发变异体信息.图 7 展示了查看 ThreadID 程序并发变异体的界面,点击具体变异体名,界面会显示源程序信息以及变异体中被修改的位置信息.另外 CMuJava 还集成了 MuJava 执行变异体的功

能,测试人员通过选择测试用例执行生成的变异体,最终得到测试结果.

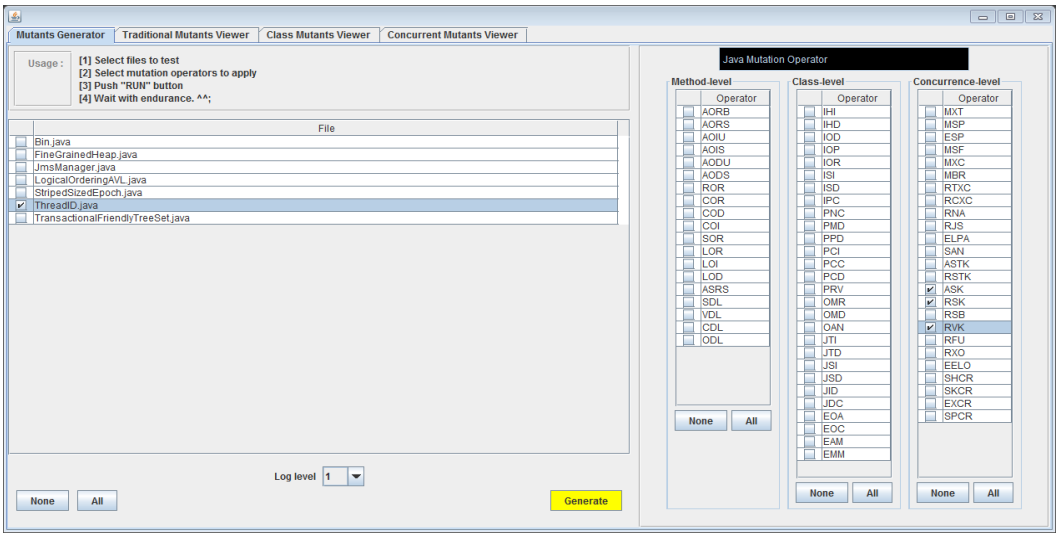

Fig.6 GUI of CMuJava

图 6 CMuJava 界面

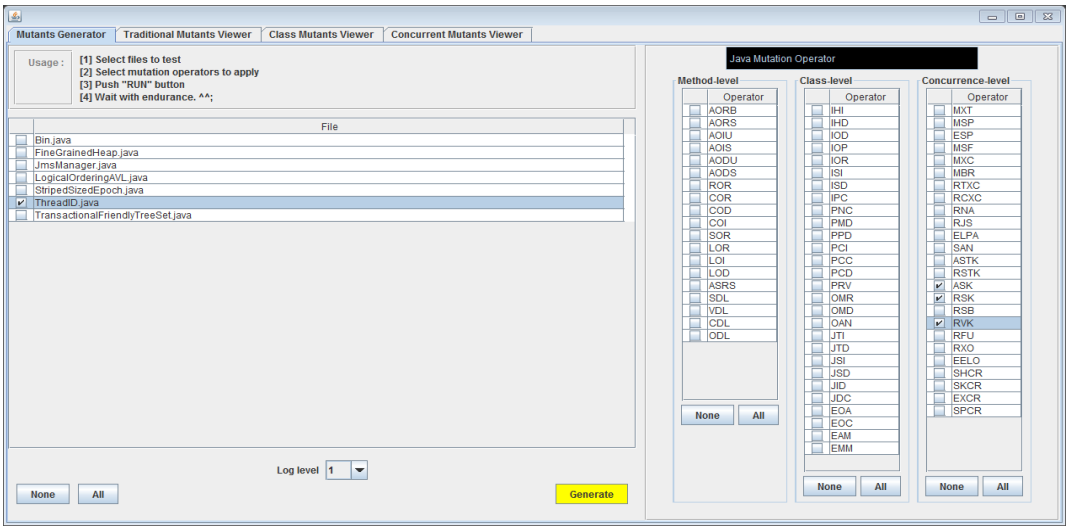

Fig.7 GUI of Concurrent Mutation Viewer

图 7 并发变异体查看界面

3 经验研究

本文选取 7 个真实的并发测试基准程序验证所提工具 CMuJava 的性能,本节对实验的设置和细节进行介绍.

3.1 研究问题

通过实例研究,我们希望回答如下三个问题.

(1) 使用 CMuJava 工具是否能够提高生成并发变异体的正确率?

使用一些被广泛使用的并发程序来进行实验,验证 CMuJava 是否能够识别程序内部所有的并发机制并生成正确的并发变异体,然后计算正确率,将实验结果和人工生成的并发变异体的正确率进行对比分析,评估使用工具是否提高了正确率.

(2) 使用 CMuJava 工具生成并发变异体是否能够提高生成的并发变异体的完备性?

首先分析并发程序内部所使用的并发机制确定程序理论上应该生成的并发变异体的数量,将工具生成的变异体和人工生成的变异体数量分别与理论数量进行对比分析,评估工具生成变异体是否提高了完备性.

(3) 使用 CMuJava 工具是否能够提高生成并发变异体的效率?

记录工具生成所有并发变异体所需要的时间,然后将实验结果和人工生成并发变异体需要的时间进行对比分析,评估工具的运行效率.

3.2 实验对象

为了让程序的规模尽可能的与现实情况比较接近,我们选取了覆盖了从几十行到上千行的不同复杂度的 7 个并发程序:Bin、ThreadID(TID)、FineGrainedHeap(FGH)、StripedSizedEpoch(SSE)、Logi-calOrderingAVL(LOAVL)、TranscationalFriendlyTreeSet(TFTS)和 JmsManager(JM).7 个并发测试基准程序涵盖了 Java 语言的常用并发机制,不同的并发程序使用的并发机制也不完全相同.表 2 列出了每个实验对象的名字、代码行数以及可应用的并发变异算子.程序 1~3 是选自文献[33]的三个并发测试基准程序.程序 4~6 来自于一个专门的并发测试基准程序库 Synchrobench[34],程序 7 来自于 Apache 开源库下的日志组件 Log4j[1].

Table 2 The information of experimental object

表 2 实验对象信息

编号	程序名	代码行数	可应用的并发变异算子
1	Bin	42	RSK
2	TID	50	ASK, RSK, RVK
3	FGH	225	ELPA, RCXC
4	SSE	148	MSP, RTXC, RNA, RSB, RVK
5	LOAVL	1088	EELO, ELPA, RCXC, RVK, RXO
6	TFTS	617	ASK, RJS, RSK, RVK
7	JM	490	ASK, RVK, MSP, SHCR, SPCR SKCR, EXCR

3.3 度量指标

针对前文提出的研究问题,本次经验研究的自变量为提出的并发变异体生成工具 CMuJava 生成变异体和测试人员人工生成变异体.其中 CMuJava 为实验组,测试人员为对照组.两实验组分别对 7 个程序进行变异体生成实验.

本文采用正确率指标来评估生成并发变异体的正确性,其计算公式如式 5 所示,

$$\text{正确率} = \frac{\text{生成的正确的并发变异体}}{\text{生成的所有的变异体}} \times 100\% \qquad (5).$$

通过统计生成并发变异体的数量来评估生成并发变异体的完备性,根据实验结果计算 CMuJava 和测试

[1] https://logging.apache.org/log4j/2.x/

人员人工生成的所有正确的并发变异体数量,并与待测程序理论上可能生成的并发变异体数量进行对比,判断两种方式是否能够正确地识别出程序里所有的可变异点并生成相应的并发变异体.

通过记录 CMuJava 解析待测程序和生成并发变异体的时间,然后对比人工对相同程序进行并发变异体生成实验所需要的时间,来评估 CMuJava 生成并发变异体的效率.

3.4 实验设置

1. 实验环境

CMuJava 运行在 64 位 Windows 10 操作系统,Java 环境为 JDK 1.8 主要配置为:Intel Core i7-7700HQ CPU、16GB 运行内存.测试人员手动生成并发变异体时所使用的集成开发环境为 Eclipse.

2. 实验步骤

本文实验基本流程如下:

(1) 从实验对象中选取一个待测程序,然后开始计时.

(2) 测试人员需要先阅读程序,了解程序的基本功能和程序所使用的并发机制.CMuJava 需要读取待测程序并进行解析,生成待测程序的元对象.

(3) 测试人员需要理解并发变异算子所定义的规则和要求,然后根据并发变异算子寻找待测并发程序里是否存在相应的可变异点.CMuJava 也需要根据并发变异算子的规则对待测程序所生成的元对象进行遍历,寻找符合并发变异算子变异规则的可变异点.如果找到相应的可变异点则进行步骤(4).否则继续根据下一个并发变异算子的规则进行可变异点查找直至完成所有并发变异算子的判断.

(4) 按照并发变异算子所定义的规则对识别的可变异点进行语法变化生成对应的并发变异体.

(5) 判断是否已生成当前程序的所有并发变异体,如果没有则继续生成下一个并发变异体.如果生成完毕则进入下一步.

(6) 停止计时,计算 CMuJava 和测试人员完成该待测程序的并发变异体生成工作所使用的时间,同时分别统计 CMuJava 和测试人员生成正确并发变异体的数量.

(7) 判断是否所有的待测程序都已经完成生成并发变异体的工作,如果没有完成,则回到步骤(1),继续选取下一个待测程序进行实验.如果已经完成,则分别计算 CMuJava 和测试人员所使用的平均时间.

在完成以上所有步骤后可以得到测试人员和 CMuJava 生成并发变异体的实验数据,然后对实验数据进行对比分析,评估使用 CMuJava 对于提高生成并发变异体的正确率、完备性和效率的作用.

3.5 有效性威胁

采用对比实验的方法验证工具的有效性,同时为了保证数据的正确性和代表性,采用多次试验并求取平均值的方法,消除了由于实验数据不准确造成的外部有效性威胁,使实验结果更具说服力.

另一个重要的外部有效性威胁是实验对象的代表性问题.我们选取的 7 个常用的并发程序作为实验对象,其中,Bin、ThreadID 和 FineGrainedHeap 三个程序,课题组前期已经应用其做了大量相关实验,能够保证程序的正确性和代表性;StripedSizedEpoch、Logi-calOrderingAVL、TranscationalFriendlyTreeSet 三个程序选自专门的并发测试基准程序库 Synchrobench,也可以保证代表性.

4 实验结果与分析

针对 7 个实验对象,实验组和对照组分别进行了 5 次实验.CMuJava 在设计的时候添加了语法检查功能,如果生成的变异体无法通过语法检查,则不会被输出到文件系统中.所以 5 次实验 CMuJava 生成变异体的正确率都为 100%.由于不同测试人员的能力差异和对变异算子理解的差异,部分测试人员会生成了一些语法

错误的变异体.图 8 显示了针对每个实验对象的对比实验中 CMuJava 生成的并发变异体正确率比人工生成的正确率高的次数(CMuJava Wins)及两种方式生成并发变异体正确率相等的次数(测试人员=CMuJava)的统计结果.大多数程序两种方式生成的并发变异体正确率相等,而对于一些程序(如,ThreadID、FineGrainedHeap、TranscationalFriendlyTreeSet),人工生成并发变异体时由于一些原因导致变异体正确率较低.从图中可以看出,使用工具对提升生成并发变异体的正确率有一定的帮助,尤其是针对一些比较特殊的并发程序(如 ThreadID),正确率的提高程度更为显著.

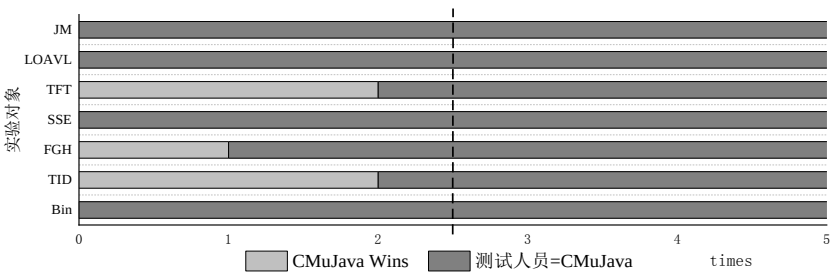

Fig.8 Correct rate of concurrent mutants

图 8 并发变异体正确率

图 9 显示了针对不同实验对象,测试人员和 CMuJava 两种方式生成并发变异体数量的相对差异百分比(我们在图中标出了两种方式下多次实验的平均值).在 7 个实验对象中,对于简单程序(如,Bin 程序)测试人员和 CMuJava 均能够正确识别程序中的并发机制并生成所有的并发变异体,但是随着待测并发程序代码复杂度和可应用的并发变异算子数量的增加,人工生成方式遗漏的并发变异体数量也越来越多,而 CMuJava 依然能够正确识别程序中的所有可变异点并生成相应的并发变异体.即使用工具提高了生成并发变异体的完备性.

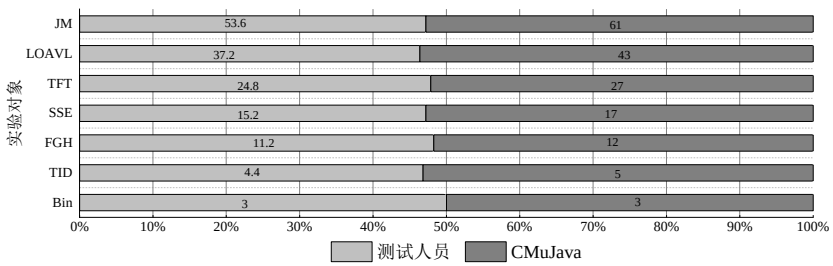

Fig.9 Number of concurrent mutants

图 9 并发变异体数量

为了评估工具的运行效率,我们统计了两种方式下生成并发变异体所用的时间.表 3 列出了 5 个测试人员和 CMuJava 针对不同待测程序 5 次实验的时间数据.可以看出随着程序复杂度的增加,测试人员和工具所需要的时间均随之增加.另外从表中可以明显看出 CMuJava 提高了生成并发变异体的效率,针对所有的程序,人工生成并发变异体所需要的时间都是 CMuJava 的上千倍.为了说明工具生成并发变异体的稳定性,我们取表中两组实验数据的平均值绘制了折线图,如图 10.从图中可以看出(1)随着程序的复杂度和可生成的并发变异体数量增加,测试人员和 CMuJava 的耗时差距也越来越大;(2)由于待测程序的复杂程度不同,人工方式生成不同程序的并发变异体所需要的时间差距很大,而 CMuJava 生成不同程序的变异体所用时间差距明显较小.因此 CMuJava 具有较高的运行效率和稳定性.

Table 3 Time of generating concurrent mutants(second)
表 3 生成并发变异体用时(秒)

生成变异体的对象	生成并发变异体所用时间						
	Bin	TID	FGH	SSE	TFT	LOAVL	JM
tester1	512	555	1004	1638	2903	4553	6765
CMuJava	0.063	0.077	0.140	0.250	0.581	0.984	1.187
tester2	523	545	944	1701	3251	4452	7219
CMuJava	0.070	0.077	0.156	0.234	0.503	0.921	1.321
tester3	448	508	900	1534	2953	4353	6891
CMuJava	0.054	0.079	0.098	0.250	0.519	0.962	1.191
tester4	487	540	893	1758	3361	4616	6881
CMuJava	0.052	0.088	0.120	0.232	0.499	0.906	1.201
tester5	524	582	987	1880	3097	4520	6098
CMuJava	0.062	0.090	0.109	0.249	0.523	0.895	1.296

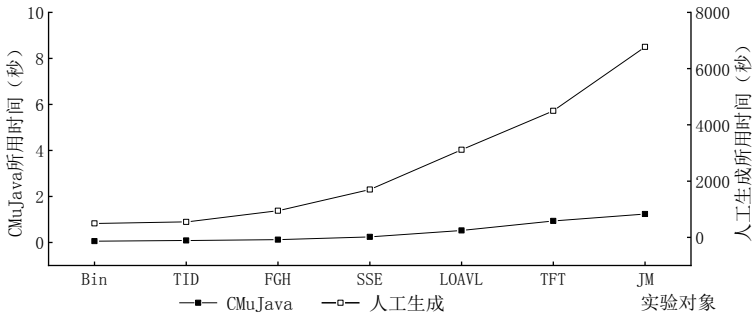

Fig.10 comparison diagram of the time generating concurrent mutants
图 10 生成并发变异体用时对比图

经验研究结果表明,使用 CMuJava 可以显著地提高面向 Java 应用程序的并发变异测试自动化程度,降低测试资源的开销并提高生成并发变异体的效率.在生产并发变异体的正确率方面,研究结果表明,相比于人工生成方式使用 CMuJava 可以将平均正确率提高约 4%.对于一些容易出错的并发变异体(例如 RCXC 和 ASK)提高的比例会更高;在生成并发变异体的完备性方面,使用 CMuJava 可以有效地避免人工生成并发变异体方式遗漏变异体的问题,保证能够准确识别待测程序的所有可变异点并生成相应的并发变异体,经验研究表明,相比于人工生成方式,使用 CMuJava 可以将生成并发变异体的数量提高约 10%;在生成并发变异体效率方面,从经验研究结果可以看出,相比于人工生成并发变异体的方式,使用 CMuJava 可以将生成效率最高提高约 8000 倍.因此,基于经验研究结果,相比于人工生成并发变异体,使用 CmuJava 工具可以显著地提高生成并发变异体的正确性、完备性和效率.

5 相关工作

自变异测试的方法被提出以来,测试人员在该领域做了大量的研究工作,涉及 C/C++、C#、Java、SQL 等多种编程语言[35].其中一些方面的工作为针对不同编程语言的变异算子的定义[20,36,37,38,39,40,41]、针对不同编程语言的变异体自动化生成工具[5,42,43],变异算子的选择[9,10],以及变异测试技术优化[23].本节介绍与本文提出的面向 Java 程序的并发变异体生成工具相关的工作.

在面向 Java 程序的并发变异算子研究方面,Delamaro 等人根据 Java 语言的并发机制设计了 15 个并发变异算子[37],并且根据这些并发变异算子所针对的变异对象不同,将这些并发变异算子分成了针对监听对象、针对并发方法调用和针对等待集合三类.Farchi 等人[38]根据 Java 语言并发机制的特点并结合开发人员在实际编码中常犯的一些并发错误,系统地归纳并总结出了一套并发故障模式.Bradbury 等人[20]对 Farchi 等人提出的并发故障模式进行了补充,然后以这些并发故障模式为依据,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、交换并发对象和修改临界区等五类共 25 种并发变异算子.Wu 和 Kaiser[39]认为已有的并发变异算子无法生成某些并发变异体,对已有的变异算子进行组合得到了 6 种新的并发变异算子,将这些并发变异算子分成了同步方法和同步代码块两类.Gligoric 等人[10]在实验过程中提出了 3 种新的并发变异算子,并结合 Bradbury 等人提出的 25 种并发变异算子一起应用到实验中.

变异算子是变异测试的一个核心概念,依据变异算子生成变异体在变异测试过程中至关重要.考虑到程序的复杂性,人工生成变异体会耗费大量的人力物力,测试人员需要一个自动生成变异体的工具来提高测试执行效率.随着变异算子的发展,大量针对不同编程语言的自动生成变异体的工具被实现,Papadakis 等人对已有的变异测试工具做出了总结[35].其中包括面向 Java 语言的变异体生成工具[5,6,7,8].MuJava 是一个面向 Java 程序的变异测试工具,实现了传统的方法级别和类级别的变异算子[5,36],但是并不支持并发变异体的生成.Javalance 为 Java 程序提供了开源的变异测试框架,但同样不支持并发变异体的生成.Jumble 是一个变异 Java 程序字节码的工具,提供了传统的方法级别的变异,并支持使用多线程程序的运行,但是文中没有提到相关的实现细节.目前已有的面向并发程序的变异工具[9,23,44]仍存在一些问题.Paraμ[9]是一个支持并发变异体生成的工具,但是该工具的研究仍处于初级阶段,仅实现了 2 个类级别变异算子和 3 个并发变异算子.MutMut[23]是一个面向并发程序的优化变异体执行的工具,据报道该工具对变异测试的时间降幅达 77%,但 MutMut 并不具有优化变异体生成的功能.另一个面向 Java 程序的支持并发变异体生成的工具 Comutation[10]实现了所有的 28 种并发变异算子,但是由于 Intel 没有对该工具授权发布,Comutation 目前并不支持开源使用.

6 总结

并发变异测试将变异测试应用到并发程序测试中,可以评估测试用例集的充分性与测试技术的有效性.由于缺少面向 Java 程序的开源的并发变异测试工具,研究者通常采用需要消耗大量测试资源的人工方式生成并发变异体.本文根据 Bradbury 等人提出的 Java 并发变异算子[7],设计并实现了一个面向 Java 程序的并发变异体自动生成工具 CMuJava,该工具扩展了传统的变异测试工具 MuJava,支持待测程序选择、变异算子选择、并发变异体生成、并发变异体查看、变异体执行和测试报告打印等功能,提高了并发变异测试的自动化程度.另外选择 7 个并发程序评估了 CMuJava 生成并发变异体的性能.实验结果表明:使用 CMuJava 可以显著提高生成并发变异体的正确性、完备性和效率.

未来,我们将在如下几个方面进一步完善我们在面向 Java 程序的并发变异体生成工具方面的研究:(1)本文使用了 7 个真实的并发程序进行经验研究,研究对象数量较少,未来需要使用更多的实验对象评估本文提出的支持工具的实用性.(2)完善 CMuJava 的功能和性能.CMuJava 目前还只是一个用于并发变异测试研究的原型工具,仍存在不足之处,例如界面设计不够友好.未来需要在此原型工具的基础上进行改进,让 CMuJava 更好地运用到并发变异测试研究领域,提高面向 Java 程序的并发变异测试技术的自动化程度.

References:

[1] Bianchi FA, Margara A, Pezzè M. A survey of recent trends in testing concurrent software systems. IEEE Transaction on Software Engineering. 2018, 44(8): 747-783.

[2] Demillo RA, Lipton RJ, Sayward FG. Hints on test data selection: Help for the practicing programmer. IEEE Computer, 1978, 11(4): 34-41.

[3] Chen X, Gu Q. Mutation testing: Principal, optimization and application. Journal of Frontiers of Computer Science and Technology, 2012, 6(12): 1057-1075 (in Chinese with English abstract).

[4] Carver R. Mutation-based testing of concurrent programs. Proceedings of International Test Conference (ITC 1993). IEEE Computer Society, 1993: 845-853.

[5] Ma YS, Offutt AJ, Kwon YR. MuJava: An automated class mutation system. Software Testing Verification and Reliability, 2005, 15(2): 97-133.

[6] Schuler D, Zeller A. Javalanche: Efficient mutation testing for java. 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.

[7] Irvine SA, Pavlinic T, Trigg L, Cleary JG, Inglis S, Utting M. Jumble java byte code to measure the effectiveness of unit tests. Proceedings of the Testing: Academic & Industrial Conference Practice & Research Techniques-mutation. IEEE Computer Society. 2007: 169-175.

[8] René J, Schweiggert F, Kapfhammer GM. Major: An efficient and extensible tool for mutation analysis in a java compiler. Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). ACM, 2011, 1: 612-615.

[9] Madiraju P, Namin AS. Paraμ: A partial and higher-order nutation tool with vonvurrency operators. Proceedings of IEEE 4th Internation Conference on Software Testing. IEEE Press, 2011: 351-356.

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

[11] Kintis M, Papadakis M, Papadopoulos A, Valvis E, Malevris N. Analysing and comparing the effectiveness of mutation testing tools: A manual study. Proceedings of IEEE 16th International Working Conference on Source Code Analysis and Manipulation. IEEE, 2016, 1: 147-156.

[12] Saifan AA, Alazzam I, Akour M, Hanandeh F. Regression test-slection technique using component model based modification: Code to test traceability. International Journal of Advanced Computer Science and Applications, 2016, 7(4): 88-92.

[13] Masood A, Bhatti R, Ghafoor A, Mathur A. Scalable and effective test generation for role-based access control systems. IEEE Transaction on Software Engineering, 2009, 35(5): 654-668.

[14] Sun CA, Wang G, Cai KY, Chen TY. Distribution-aware mutation analysis. Proceedings of IEEE Computer Software & Applications Conference Workshops. IEEE Computer Society, 2012: 170-175.

[15] Sun CA, Wang G. MuJavaX：A distribution-aware mutation generation system for java. Journal of Computer Research and Development, 2014, 51(4): 874-881 (in Chinese with English abstract).

[16] Kim SW, Ma YS, Kwon YR. Combining weak and strong mutation for a noninterpretive java mutation system. Software Testing, Verification and Reliability. 2013, 23(8): 647-668.

[17] Lipton RJ. Reduction: A method of proving properties of parallel programs. Communications of the ACM, 1975,18(12): 717–721.

[18] Lamport L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 1978, 21(7): 558-565.

[19] Morell LJ. A theory of fault based testing. IEEE Transaction on Software Engineering. 1990, 16(8): 844-857.

[20] Bradbury JS, Cordy JR, Dingel J. Mutation operators for concurrent java (J2SE 5.0). Proceedings of the 2th Workshop on Mutation Analysis. IEEE Computer Society, 2006: 83-92.

[21] Farchi E, Nir Y, Ur S. Concurrent bug patterns and how to test them. Proceedings of the 17th International Symposium on Parallel & Distributed Processing (IPDPS 2003). IEEE Computer Society, 2003: 286.2.

[22] Bradbury JS, Cordy JR, Dingel J. Mutation operators for concurrent java (J2SE 5.0). Technical report 2006-520, Queen's University, 2006.

[23] Gligoric M, Jagannath V, Marinov D. MutMut: Efficient exploration for mutation testing of multithreaded code. Proceedings of the third International Conference on Software Testing. IEEE Press, 2010: 55-64.

[24] Kiczales G, Rivieres J, Bobrow DG. The art of the metaobject protocol. MIT Press, 1991.

[25] Kim S, Clark J, McDermid J. Assessing test set adequacy for object oriented programs using class mutation. Proceedings of Symposium on Software Technology (SoST 1999). 1999, 72-83.

[26] Untch RH, Offutt AJ, Harrold MJ. Mutation analysis using mutant schemata. Proceedings of ACM Sigsoft International Symposium on Software Testing and Analysis (ISSTA 1993). ACM Press, 1993: 139-148.

[27] Chevalley P. Applying mutation analysis for Object-oriented programs using a reflective approach. Proceedings of the 8th Asia-Pacific Software Engineering Conference (APSEC 2001). IEEE Computer Society, 2001, 1: 267-270.

[28] Tatsubori M, Chiba S, Killijian M O, Itano K. OpenJava: A class-based macro system for Java. Reflection & Software Engineering, 2000, 1826(1): 117-133.

[29] Chiba S. Load-time structural reflection in Java, Ecoop, 2000: 313-336.

[30] Kleinoder J, Golm M. MetaJava: An efficient run-time meta architecture for Java. Proceedings of the 5th International Workshop on Object-Orientation in Operating Systems. IEEE, 1996: 54-61.

[31] Welch I, Stroud RJ. Kava: Using byte code rewriting to add behavioral reflection to Java. Technical report 704, University of Newcastle upon Tyne, 2000.

[32] Pati T, Hill JH. A survey report of enhancements to the visitor software design pattern. Software: Practice and Experience, 2014, 44(6): 699-733.

[33] Herlihy M, Shavit N. The art of multiprocessor programming. Morgan Kaufmann, 2012.

[34] Gramoli V. More than you ever wanted to know about synchronization synchrobench, measuring the impact of the synchronization on concurrent algorithms. Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM Press, 2015: 1-10.

[35] Papadakis M, Kintis M, Zhang J, Jia Y, Traon YL, Harman M. Mutation testing advances: An analysis and survey. Advances In Computer, 2019: 275-378.

[36] Offutt J, Ma YS, Kwon YR. The class-level mutants of MuJava. Proceedings of International Workshop on Automation of Software Test. ACM press, 2006: 78-84.

[37] Delamaro M, Pezzè M, Vincenzi A, Maldonado JC. Mutant operators for testing concurrent java programs. Proceedings of the Brazilian Symposium on Software Engineering (SBES 2001). IEEE Computer Society, 2001: 272-285.

[38] Farchi E, Nir Y, Ur S. Concurrent bug patterns and how to test them. Proceedings of the International Parallel and Distributed Processing Symposium (PDPS 2003). IEEE Computer Society, 2003: 286-292.

[39] Wu L, Kaiser G. Constructing subtle concurrency bugs using synchronization-centric second-order mutation operators. Proceedings of the 23th International Conference on Software Engineering and Knowledge Engineering (SEKE 2011). 2011: 244-249.

[40] Ma YS, Kwon YR, Offutt J. Inter-class mutation operators for java. Proceedings of International Symposium on Software Reliability Engineering. IEEE, 2002: 352-366.

[41] Wu YB, Guo JX, Li Z, Zhao RL. Mutation strategy based on concurrent program data racing fault. Joumal of Computer Application, 2016(11): 3170-3177, 3195 (in Chinese with English abstract).

[42] Delgado-Pérez P, Medina-Bulo I, Palomo-Lozano F, García-Domínguez A, Domínguez-Jiménez JJ. Assessment of class mutation operators for C++ with the mucpp mutation system. Information and Software Technology, 2017, 81: 169–184.

[43] Kusao M, Wanng C. Ccmutator: A mutation generator for concurrency constructs in multithreaded C/C++ applications. Proceedings of the 28th IEEE/ACM International Conference on Automated Softwafre Engineering. 2013: 722-725.

[44] Bradbury JS, Cordy JR, Dingel J. Comparative asscessment of testing and model checking using program mutation. Proceedings of Testing: Academic & Industrial Conference Practice & Research Techniques-mutation. IEEE Computer Society, 2007: 210-219.

附中文参考文献:

[3] 陈翔,顾庆.变异测试:原理、优化、和应用.计算机科学与探索,2012,6(12):1057-1075.

[15] 孙昌爱,王冠.MuJavaX:一个支持非均匀分布的变异生成系统.计算机研究与发展,2014,51(4):874-881.

[41] 吴俞伯,郭俊霞,李征,赵瑞莲.基于并发程序数据竞争故障的变异策略.计算机应用,2016,36(11):3170-3177,3195.