|  |
| --- |
| **收 件 记 录** |
| **日期：**  **收件人：** |

**中国科学院软件研究所**

**计算机科学国家重点实验室开放课题基金**

**申 请 书**

**申请人姓名：**

**￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣**

**所在单位：**

**￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣**

**申请课题：** 面向并发程序的蜕变测试技术研究

**￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣**

**申请金额：      元人民币**

**￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣**

**年限起止：**      **年     月 ～**      **年     月**

¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

**填报日期：      年     月     日**

**￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣**

**中国科学院软件研究所**

**二零一一年制**

**填 报 说 明**

1. 填写申请书前，请先阅读“开放课题基金管理办法”和“开放课题基金申请指南”。表格的各项内容，要实事求是，逐条认真填写。表达要明确、严谨，字迹要清晰。外来语要同时用中文和原文表达。第一次出现的缩写词，须注出全称。
2. 请用A4纸以正反面打印，并于左侧装订成册。各栏空格不够时，请自行加页。
3. 申请书一式三份，由所在单位学术委员会签署意见后，投送计算机科学重点实验室。
4. 中级科研人员申请开放课题，需有高级职称的科研人员作为推荐人。
5. 通讯地址：北京市海淀区中关村南四街四号，中国科学院软件研究所，计算机科学国家重点实验室，邮政编码：100190。来函请注明“开放课题”。
6. 办公电话：010-62661616

传真号码：010-62661627 Email地址：zli@ios.ac.cn。

**申请人及课题组情况**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **申请人** | 姓名 |  | | | | 性别 |  | | | 出生年月 | | |  | | | | | |
| 民族 |  | | | | 职称 |  | | | 身份证号码 | | | | |  | | | |
| 通信地址、邮编 |  | | | | | | | | | | | | | | | | |
| 联系电话 |  | | | | | | | 电子邮箱 | | |  | | | | | | |
| **项目组** | 总人数 | 高级 | | 中级 | | | | 初级 | | | 博士后 | | | 博士生 | | 硕士生 | | 其他 |
|  |  | |  | | | |  | | |  | | |  | |  | |  |
| **成员姓名** | 职称/学位 | 性别 | 年龄 | | 身份证号码 | | | | | | 工 作 单 位 | | | | | | 签名 | |
|  |  |  |  | |  | | | | | |  | | | | | |  | |
|  |  |  |  | |  | | | | | |  | | | | | |  | |
|  |  |  |  | |  | | | | | |  | | | | | |  | |
|  |  |  |  | |  | | | | | |  | | | | | |  | |
|  |  |  |  | |  | | | | | |  | | | | | |  | |
|  |  |  |  | |  | | | | | |  | | | | | |  | |
|  |  |  |  | |  | | | | | |  | | | | | |  | |

**一、申请者正在承担的其它研究项目情况**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 项目名称 | 起始日期 | 负责或参加 | 来 源 | 进展或完成情况 |
|  |  |  |  |  |

**二、申请人主要学术经历**（包括大学以上学历，工作经历及论文、成果等）

|  |
| --- |
|  |

**申 请 课 题 介 绍**

1. **研究的意义、预期成果、应用前景及国内外发展综述**

|  |
| --- |
| 1. **研究的意义及国内外发展综述**   近年来，多核计算日益普及，并发程序广泛用来提升此类计算系统的资源利用效率。并发程序中存在多个并发执行的流程，这些流程之间通常显式或隐式地共享一些存储空间，并且它们执行的次序不确定[1]。通常，并发流程之间不确定相互作用与影响的情形称为执行交错(Inter-leaving)。执行交错不仅易于导致并发程序在设计阶段引入新的缺陷，而且给并发程序的后期分析与测试带来了挑战。**如何有效地检测并发程序中潜藏的缺陷、提高并发程序的可靠性已经成为并发程序开发领域的一个亟待解决的重要问题**。  软件测试是一种广泛采用的软件质量保证手段[2]，通过运行有限的测试用例，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。由于执行交错的存在，并发程序的测试工作更具挑战性。**近年来，并发程序测试方面的研究工作比较活跃[1]，代表性研究工作包括**：   1. **并发程序的测试用例生成方面：**在软件测试中，生成测试用例是一个重要的步骤，它直接影响测试技术揭示故障的效率。目前并发程序测试技术的测试用例主要通过两种方式得到：（1）随机的方法自动生成测试用例[4]。随机地产生测试用例简单、快捷但是没有利用待测软件的信息，因此这种方法产生的测试用例可能不利于揭示难检测的并发故障；（2）在已有的测试用例集中，选出涉及到将要执行的一系列操作的测试用例，然后将这些顺序执行的测试用例构建成并发测试用例。并行程序可以在同一时间执行多个流程，因此该方式产生的测试用例可以解决并行程序的测试输入问题。 2. **并发程序执行交错方面：**流程之间执行的次序不受控制，每一种流程执行序列都是一个交错模式。**在其中一些交错模式下，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**。在并发程序中常见的交错性质有以下3种：（1）**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时，数据竞争就会出现；（2）**原子违反**。该故障类型发生在某一流程中期待具有原子性的一系列操作与来自其它流程的冲突操作交错；（3）**死锁**。该故障类型导致同步机制无限期地阻止一些流程继续执行。**根据关注执行交错的角度不同，现有的研究并发程序执行交错的技术可以分为两类**：（1）测试技术主要关注待测并发程序的执行交错是否会出现某一种交错性质，这类技术称之为**基于交错性质的技术**。这类技术主要通过分析一个执行轨迹识别指令间同步约束关系，然后判断当前的执行交错是否符合某一种交错性质或者由此执行交错推断是否存在可能出现某种交错性质的执行交错；（2）有的测试技术不关注某一个或者某些交错性质，探索待测并发程序中可能出现的所有交错模式组成的交错空间，这类技术称之为**交错空间探索技术**。这类技术根据探索交错空间的方式不同主要可以分为3种：（a）**压力测试技术**。传统的压力测试是在不断加重的负载下运行目标系统来确定目标系统的性能瓶颈。在并发程序种，压力测试反复执行同一个测试用例集，旨在观察可能出现的执行交错。由于测试用例是固定的，这种技术对交错空间的探索是有限的；（b）**穷尽探索技术**。这类技术旨在执行待测并发程序中所有的交错模式。随着程序规模扩大和流程数目的增加，交错模式的数目呈指数式增长。因此这类技术要么同时限定程序的规模与流程的数目，要么限制交错空间的大小；（c）**覆盖标准**。这类技术主要利用待测并发程序的内部信息识别交错空间的一个子集。   **接下来介绍每一种技术的主要研究工作**，**首先基于交错性质的研究工作如下：**   1. **数据竞争**。检测数据竞争的并发测试技术通常先分析给定执行流程中访问共享内存指令间的执行次序，然后判断是否发生了数据竞争或者预测可能出现数据竞争的执行交错。在并发程序中流程执行一个代码片段或者执行某一个指令称为事件（event），因此每一个流程包含多个事件。为了表示并发程序中多个流程中不同事件的执行次序，Lamport 等人提出happens-before 分析方法对程序中的事件进行排序[5]，最后分析排序后的事件是否可能发生数据竞争。由于该方法需要每个执行流的信息，以及所有的执行流对每个共享数据项的访问信息，该方法需要大量的时间开销。Flanagan 等人通过保存额外的运行信息提高了happens-before方法的性能[6]。Marino 等人提出一个轻量级的探测器——LiteRace [7]，该探测器采样和分析程序中被选择的部分而不是整个程序从而降低了时间开销。Wester 等人将数据竞争探测工作分散到多个计算核心中，进一步加快了happens-before方法的测试效率[8]。**基于共享变量经常在没有持有锁的情况下被初始化这一观察**，Savage 等人提出锁集分析的方法检测数据竞争[9]。该方法为每一个待测并发程序的共享变量创建一个锁集合，该集合初始时包含所有可能的锁对象。当流程访问 时，将 与当前流程持有锁形成的集合的交集作为新的。如果，则表明共享变量 没有被锁保护，流程之间可能出现数据竞争。Shacham等人将动态锁集分析与模型检测结合来揭示一般共享内存程序的数据竞争故障[10]。该方法首先通过动态锁集分析捕获锁约束信息，然后利用模型检测产生可选择的执行交错，最后执行识别的交错来揭示故障。Sheng等人利用锁集分析技术开发Racez工具来检测并发系统的数据竞争故障[11]。该工具只捕获目标系统的同步操作和内存访问的一个子集，从而降低锁集分析技术的时间开销。 2. **原子违反。**许多检测原子违反的并发测试技术利用执行交错串行化验证程序的正确性。根据定义具有原子性的区域不同，可以将这类技术分为2类：(a) **以代码为中心的原子违反技术。**该类技术**根据软件的规格说明书确定具有原子性的代码区域，然后在识别的代码区中检测是否可能出现违反原子性的执行交错**。一个方法（或一个代码块）具有原子性当且仅当这个方法在执行过程中不被其它流程的事件交错。Flanagan等人基于这一理论提出了Atomizer方法[12]，该方法首先通过执行待测程序得到并发程序的事件序列，然后根据定义的规则移动事件发生的次序。如果最终某一个方法（或一个代码块）的所有事件没有被其它流程的事件交错则没有出现原子违反，反之则出现。许多检测原子违反的工作都是在Atomizer方法基础上进行改进的。Lut提出了AVIO方法[13]，该方法根据待测程序的结构以及执行信息推断原子代码区。Chen 等人将静态分析与动态分析结合从而减少了假阳性故障[14]。Wang等人利用面向对象程序的特征优化了Atomizer方法[15]；(b) **以数据为中心的原子违反技术。**这类技术开始于Vaziri等人提出的atomic-set 线性化概念[52]。该概念建立在atomic set与unit of wrok基础上，前者表示一些变量的集合，并且这些变量的更新操作具有原子性，后者代表更新atomic set中元素的操作。这类技术主要涉及到发现违反原子集可线性化的实例[16]。近年来，该类技术主要聚焦在：(a) 分析数据项之间的相关性[17]；(b) 揭示数据访问模式[18][19]；(c) 利用并发系统的各个版本信息进行回归测试[20]。 3. **死锁。**死锁主要原因是不正确的共享资源访问次序，例如两个线程共享两个锁对象，并且这两个线程访问这两个锁的顺序相反。基于这一观察，Havelund 等人提出GoodLock运行时分析算法[21]。该算法在程序运行时记录每一个流程访问不同锁对象的次序构建锁树，在程序终止后成对的比较锁树并判断对应的执行流之间是否可能出现死锁。由于锁树忽略了程序的细节只记录每一个流程访问锁对象的次序，因此这种技术可能出现很多假阳性故障。Joshi等人在GoodLock方法的基础上提出了DeadLockFuzzer的动态分析方法[22]。该方法分两步检测程序中可能存在的死锁：首先构建锁树，并分析程序中可能潜在的死锁，然后执行预测造成死锁的交错。这种方法可以有效降低Good Lock方法产生的假阳性故障。Samak 等人利用GoodLock方法构建锁树，然后进行修剪从而提高了GoodLock的性能[23]。Samak等人利用顺序执行的属性构造并发测试用例来揭示程序中可能存在的死锁[24]。   **交错空间探索技术的主要研究工作如下：**   1. **压力测试。**压力测试通过重复执行固定的测试用例集观察不同的执行交错。有关这方面的研究较少，已有的技术要么检查程序执行的序列化程度要么检查版本变化对并发系统性能的影响。Pike 等人通过比较程序输出以及执行过程的内部状态来检测一个并发程序的线性化[25]。Pradel 等人提出了SpeedGun 技术[26]，该技术自动产生一系列的测试用例，然后在不同版本的并发程序上分别执行数次产生的测试用例，最后报告两个版本性能的差异。 2. **穷尽探索技术。**并发程序交错空间的大小受执行流程的数目和程序规模影响。由于交错空间一般都很大，分析每一种可能的交错代价很大。测试人员可以通过控制测试用例数目减小要探索的交错空间。例如，Nistor等人提出了Ballerina技术[27]，该技术仅使用两个线程并且每一个线程执行一个方法。该方法可以高效的揭示软件中的并发故障。Pradel等人认为系统崩溃或死锁也可以作为一种测试预期[28]，由此扩展了Ballerina方法。另一方面，通过符号执行或者模型检测的方法探索整个交错空间。例如，Sen使用符号执行来生成可能导致交替行为的输入数据[29]，同时使用具体的执行来确定在运行时程序执行中的部分事件顺序。这种方法通过避免许多等效行为的测试来提高算法的效率。 3. **覆盖标准。**覆盖标准利用共享内存中的数据流关系、同步代码块的访问次序或者并行系统的消息传递次序来限制要探索的程序代码量。Wang 等人提出了基于覆盖的测试框架[32]，该框架使用动态学习的共享对象访问排序约束选择高风险的交错。Hong 等人首先估计synchronization-pair[33]的覆盖需求，然后产生线程调度尽可能的覆盖没有覆盖到的交错。 4. **并发程序的测试结果判定：**并发测试技术涉及到两种类型的输出：(1) 符合预期的交错性质；(2) 执行某种交错产生的最终结果。考虑到第一种输出形式，并非所有呈现预期性质的交错都会导致一个故障。因此该技术可能出现大量的假阳性故障。第二种输出结果通过与测试预期比较来判断当前的执行是失败的执行还是成功的执行。   **目前大多数并发测试技术均假设待测程序存在测试预期**。实际上，很多情况下由于并发程序执行的不确定性，待测程序的测试预期很难得到或者需要很大的代价才能运用。近几十年来，很多研究致力于缓解测试预期问题，例如：N-version测试[35]、断言[36]、机器学习[37]、假设检验[38]、蜕变测试[39]。其中蜕变测试技术从被提出以来一直受到广泛的关注。该技术从待测软件中提取蜕变关系，然后原始测试用例（可以从其它测试用例产生技术得到）根据识别的蜕变关系可以生成衍生测试用例，接着将原始测试用例和衍生测试用例分别在待测程序上执行，最后检测输出的结果是否违反蜕变关系。如果违反了蜕变关系则说明软件中存在故障。由此可以看出，在蜕变测试技术中原始测试用例与蜕变关系对蜕变测试的效率有决定性的作用。**在蜕变测试的原始测试用例方面**：Gotlieb等人提出了AMT框架[43]，该框架试图寻找能够违反蜕变关系的测试用例。Chen等人比较特殊值与随机值对蜕变测试效率的影响[44]，作者认为随机测试可以为蜕变测试提供大量的测试用例，特殊值技术产生的测试用例可以补充随机测试产生的测试用例集。Batra等人提出了遗传算法[45]，该算法挑选的测试用例可以最大化覆盖程序路径。Dong等人利用符号执行的方式提取蜕变关系并产生相应的测试用例[46]；**在提取蜕变关系方面：**Liu等人将已经存在的多个蜕变关系复合成一个新的蜕变关系[47]。Zhang通过反复执行待测程序，从输出与对应的输出之间提取线性或者多项式形式的蜕变关系[48]。Carzinaga等人通过利用程序中的冗余识别待测程序的测试预期[49]。Su等人提出了KABU方法[50]，该方法可以动态识别待测程序中的方法改变输入前后状态的改变，并由此提取蜕变关系。Chen提出了一个基于规格说明书的识别蜕变关系的方法METRIC[51]。该方法通过比较两个完整的测试帧识别蜕变关系。  我们课题组在蜕变测试方面做了如下工作：我们提出了面向Web 服务的蜕变测试框架并进行了经验学习[40][41]。此外我们课题组开发了面向Web服务的蜕变测试支持工具MT4WS[42]。在这些工作基础上本文提出了面向并发程序的蜕变测试技术。   1. **预期成果**   本课题研究面向并发系统的蜕变测试框架与技术、开发相应的支持工具原型，为开发  可靠的并发程序提供新型的测试理论与工具支持。具体来说，   1. **面向并发系统的蜕变测试理论、框架与支持工具方面，**发表高水平研究论文6-8篇，其中SCI或EI检索论文6篇。 2. **基于蜕变测试的并发故障检测框架与技术及实例研究：**采用多个并发系统实例演示与验证基于蜕变测试的并发故障检测框架，开发支持工具2个；提交基于蜕变测试的并发故障检测技术报告1份，相关支持工具技术报告2份。 3. **基于适应性随机测试的并发测试用例选择框架与技术及实例研究：**采用多个并发系统实例演示与验证基于适应性随机测试的并发测试用例选择框架，开发支持工具1个；提交基于适应性随机测试的并发测试用例选择技术与支持工具技术报告个1份。 4. **培养博士研究生2人，硕士研究生4-5人。** 5. **应用前景**   随着计算机硬件技术的发展以及人们对软件性能的要求不断增高，并发技术受到了  广泛的关注。并发程序相对于顺序程序具有计算资源利用率更高，响应时间短的优点。但是由于并发程序执行的不确定性，给测试工作也带来挑战。很多并发系统例如：分布式系统、Web服务、移动应用等，测试人员往往不容易得到测试预期，或者需要很大的代价运用测试预期。测试预期问题使得不容易检测的并发系统更难测试。例如，很多公司存在优先级队列的数据结构，并且往往包含大量的数据。当并发系统返回一系列指定优先级的数据时，判断系统返回数据的正确性是一个很困难的问题。本课题提出的面向并发系统的蜕变测试可以解决这种缺少测试预期的并发系统难测试的问题，保证并发系统的可靠性。  **参考文献**   1. F. Bianchi, A. Margara, M. Pezze. A Survey of Recent Trends in Testing Concurrent Software Systems. *IEEE Transactions on Software Engineering*, IEEE Compter Society, 2017, 1(2):1-2. 2. G. J. Myers, C. Sandler, T. Badgett. The Art of Software Testing. John Wiley and Sons, 2011. 3. Z. Lin, D. Marinov, H. Zhong, Y. Chen, J. Zhao. JaConTeBe: A Benchmark Suite of Real-World Java Concurrency Bugs. *Proceedings of the 31st International Conference on Automated Software Engineering (ASE’16)*, IEEE Compter Society, 2016, pp. 178-189. 4. R. Hamlet. Random Testing. *Encyclopedia of Software Engineering*. John Wiley and Sons, Inc., 2002. 5. L. Lamport. Time, Clocks, and the Ordering of Events in A Distributed System. *Communications of the ACM*, ACM Press, 1978, 21(7): 558-565. 6. C. Flanagan, S. N. Freund. FastTrack: Efficient and Precise Dynamic Race Detection. *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’09)*. ACM Press, 2009, pp. 121-133. 7. D. Marino, M. Musuvathi, S. Narayanasamy. LiteRace: Effective Sampling for Lightweight Data-Race Detection. *Proceedings of* *the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’09).* ACM Press, 2009, pp. 134-143. 8. B. Wester, D. Devecsery, P. M. Chen. Parallelizing Data Race Detection. *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’13).* ACM Press, 2013, pp. 27–38. 9. S. Savage, M. Burrows, G. Nelson. Eraser: A Dynamic Data Race Detector for Multithreaded Programs. *ACM Transactions on Computer Systems*, ACM Press, 1997, 15(4): 391-411. 10. O. Shacham, M. Sagiv, A. Schuster. Scaling Model Checking of Dataraces Using Dynamic Information. *Journal of Parallel and Distributed Computing*, Elsevier, 2007, 67(5): 536-550. 11. T. Sheng, N. Vachharajani, S. Eranian. RACEZ: A Lightweight and Non-Invasive Race Detection Tool for Production Applications. *Proceedings of the 33rd* *International Conference on Software Engineering (ICSE’11)*. ACM Press, 2011, pp. 401-410. 12. C. Flanagan, S. N. Freund. Atomizer: A Dynamic Atomicity Checker for Multithreaded Programs. *Proceedings of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL’04)*. ACM Press, 2004, pp. 256-267. 13. S. Lu, J. Tucek, F. Qin. AVIO: Detecting Atomicity Violations Via Access Interleaving Invariants. *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’06)*. ACM Press, 2006, pp. 37-48. 14. Q. Chen, L. Wang, Z. Yang. HAVE: Detecting Atomicity Violations Via Integrated Dynamic and Static Analysis. *Procedings of the 12th International Conference on Fundamental Approaches to Software Engineering (SASE’09)*. Springer, 2009, pp.425-439. 15. L. Wang, S. D. Stoller. Runtime Analysis of Atomicity for Multithreaded Programs. *IEEE Transactions on Software Engineering*, 2006, 32(2): 93-110. 16. M. Vaziri, F. Tip, J. Dolby. Associating Synchronization Constraints with Data in an Object-Oriented Language. *Proceedings of the 33rd Annual ACM SIGPLAN – SIGACT Symposium on Principles of Programming Languages (POPL’06)*. ACM Press, 2006, pp. 334-345. 17. S. Lu, S. Park, C. Hu. MUVI: Automatically Inferring Multi-Variable Access Correlations and Detecting Related Semantic and Concurrency Bugs. *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP’07)*. ACM Press, 2007, pp. 103-116. 18. Z. Lai, S. C. Cheung, W. K. Chan. Detecting Atomic-Set Serializability Violations in Multithreaded Programs Through Active Randomized Testing. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE’10)*. IEEE Compter Society, 2010, pp. 235-244. 19. C. Hammer, J. Dolby, M. Vaziri. Dynamic Detection of Atomic-Set-Serializability Violations. *Proceedins of the 30th International Conference on Software Engineering (ICSE'08)*, IEEE Compter Society, 2008, pp. 231-240. 20. V. Terragni, S. C. Cheung, C. Zhang. Recontest: Effective Regression Testing of Concurrent Programs. *Proceedings of the 37th International Conference on Software Engineering (ICSE’15)*, IEEE Compter Society, 2015, pp. 246-256. 21. K. Havelund. Using Runtime Analysis to Guide Model Checking of Java Programs. *Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification (SPIN’00), Co-located with the 22nd International Conference on Software Engineering (ICSE’00)*, Springer, 2000, pp. 245-264. 22. P. Joshi, C. S. Park, K. Sen. A Randomized Dynamic Program Analysis Technique for Detecting Real Deadlocks. *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’09)*, ACM Press, 2009, pp. 110-120. 23. M. Samak, M. K. Ramanathan. Trace Driven Dynamic Deadlock Detection and Reproduction. *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP’14)*, ACM Press, 2014, pp. 29-42. 24. M. Samak, M. K. Ramanathan. Multithreaded Test Synthesis for Deadlock Detection. *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA’14)*, ACM Press, 2014, pp. 473-489. 25. P. Fonseca, C. Li, R. Rodrigues. Finding Complex Concurrency Bugs in Large Multi-Threaded Applications. *Proceedings of the ACM SIGOPS EuroSys European Conference on Computer Systems (EuroSys’11)*, ACM Press, 2011, pp. 215-228. 26. M. Pradel, M. Huggler, T. R. Gross. Performance Regression Testing of Concurrent Classes. *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA’14)*, ACM Press, 2014, pp. 13-25. 27. A. Nistor, Q. Luo, M. Pradel. Ballerina: Automatic Generation and Clustering of Efficient Random Unit Tests for Multithreaded Code. *Proceedings of the 34th International Conference on Software Engineering (ICSE’14)*, IEEE Compter Society, 2014, pp. 727-737. 28. M. Pradel, T. R. Gross. Fully Automatic and Precise Detection of Thread Safety Violations. *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDL’12)*, ACM Press, 2012, pp. 521-530. 29. K. Sen, G. Agha. Automated Systematic Testing of Open Distributed Programs.*Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE’06)*, Springer, 2006, pp. 339-356. 30. S. Lauterburg, M. Dotta, D. Marinov. A Framework for State-Space Exploration of Java-Based Actor Programs. *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE’09)*, IEEE Computer Society, 2009, pp. 468-479. 31. W. Visser, S. Khurshid. Test Input Generation with Java PathFinder. *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA’04)*, ACM Press, 2004, pp. 97-107. 32. C. Wang, M. Said, A. Gupta. Coverage Guided Systematic Concurrency Testing. *Proceedings of the 33rd International Conference on Software Engineering (ICSE’11)*, ACM Press, 2011, pp. 221-230. 33. S. Hong, J. Ahn, S. Park. Testing Concurrent Programs to Achieve High Synchronization Coverage. *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA’12)*, ACM Press, pp. 210-220. 34. A. Farzan, P. Madhusudan, F. Sorrentino. Meta-Analysis for Atomicity Violations Under Nested Locking. *Proceedings of the 21st International Conference on Computer Aided Verification (CAV’09)*, Springer, 2009, pp. 248-262. 35. S. S. Brilliant, J. C. Knight, P. E. Ammann. On the Performance of Software Testing Using Multiple Versions. *Proceedings of the 20th international symposium on fault-tolerant computing (**FTCS’90)*, IEEE Computer Society, 1990, pp. 408–415. 36. K. Y. Sim, C. S. Low, F. C. Kuo. Eliminating Human Visual Judgment from Testing of Financial Charting Software. *Journal of Software*, 2014, 9(2): 298-312. 37. W. K. Chan, S. C. Cheung. PAT: A Pattern Classification Approach to Automatic Reference Oracles for the Testing of Mesh Simplification Programs. *Journal of Systems and Software*, 2009, 82(3), 422–434. 38. R. Guderlei, J. Mayer. Statistical Metamorphic Testing Testing Programs with Random Output by Means of Statistical Hypothesis Tests and Metamorphic Testing. *Proceedings of the 7th international conference on quality software (QSIC’07)*, IEEE Computer Society, 2007, pp. 404–409. 39. T. Y. Chen, S. C. Cheung, S. M. Yiu. Metamorphic Testing: A New Approach for Generating Next Test Cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998. 40. C. Sun, G. Wang, B. Mu. Metamorphic Testing for Web Services: Framework and A Case Study. *Proceedings of the 9th IEEE International Conference on Web Services (ICWS’11)*, IEEE Computer Society, 2011, pp. 283-290. 41. C. A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, T. Y. Chen. A Metamorphic Relation-Based Approach to Testing Web Services Without Oracles. *International Journal of Web Services Research*, 2012, 9(1): 51-73. 42. C. Sun, G. Wang, Q. Wen, D. Towey, T. Y. Chen. MT4WS: An Automated Metamorphic Testing System for Web Services. *International Journal of High Performance Computing and Networking*, 2016, 9(1): 104-115. 43. A. Gotlieb, B. Botella. Automated Metamorphic Testing. *Proceedings of the 27th Annual International Conference on Computer Software and Applications (COMPSAC’03)*, IEEE Computer Society, 2003, pp. 34-40. 44. T. Y. Chen, F. C. Kuo, Y. Liu. Metamorphic Testing and Testing with Special Values. *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD’04)*, 2004, pp. 128-134. 45. G. Batra, J. Sengupta. An Efficient Metamorphic Testing Technique Using Genetic Algorithm. *Proceedings of 5th International Conference on Information Intelligence, Systems, Technology and Management (ICISTM’11)*, Springer, 2011, pp. 180-188. 46. G. Dong, T. Guo, P. Zhang. Security Assurance with Program Path Analysis And Metamorphic Testing. *Proceedings of the 4th International Conference on Software Engineering and Service Science (ICSESS’13)*, IEEE Computer Society, 2013, pp. 193-197. 47. H. Liu, X. Liu, T. Y. Chen. A New Method for Constructing Metamorphic Relations. *Proceedings of the 12th International Conference on Quality Software (QSIC’12)*, IEEE Computer Society, 2013, pp. 59-68. 48. J. Zhang, J. Chen, D. Hao. Search-Based Inference of Polynomial Metamorphic Relations. *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE’14)*, ACM Press, 2014, pp. 701-712. 49. A. Carzaniga, A. Goffi, A. Gorla. Cross-Checking Oracles from Intrinsic Software Redundancy. *Proceedings of the 36th International Conference on Software Engineering (ICSE’14)*, ACM Press, 2014, pp. 931-942. 50. F. H. Su, J. Bell, C. Murphy. Dynamic Inference of Likely Metamorphic Properties to Support Differential Testing. *Proceedings of the 10th International Workshop on Automation of Software Test (AST’15), Co-located with the 37th IEEE International Conference on Software Engineering (ICSE’15)*, IEEE Computer Society, 2015, pp. 55-59. 51. T. Y. Chen, P. L. Poon, X. Xie. METRIC: METamorphic Relation Identification Based on the Category-Choice Framework. *Journal of Systems and Software*, 2016, 116: 177-190. 52. M. Vaziri, F. Tip, J. Dolby, Associating Synchronization Constraints with Data in an Object-Oriented Language. *ACM Sigplan Notices*, 2006, 41(1): 334-345. |

**二、研究内容、技术路线与方法**

|  |
| --- |
| 1. **研究内容**   在测试用例生成、缓解测试预期问题等已有的工作基础上，围绕如何缓解并发程序的测试预期问题、怎么快速找到能够揭示并发故障的测试用例等关键问题开展研究。主要研究内容如下：   1. **建立面向并发系统的蜕变测试框架：**现有的很多工作都假设测试的并发程序存在测试预期，但是实际中往往存在运用测试预期代价大或者不存在测试预期的情况。蜕变测试是一种可以有效缓解测试预期问题的测试技术。它的原理是利用多个测试用例执行待测程序，如果程序正确，则每组测试用例的输入之间与对应的输出之间应该满足某种关系。这种关系来自待测程序固有的属性，被称为蜕变关系。每一组测试用例包含两个测试用例：其中一个是原有的测试用例（可以由其它测试用例生成技术得到）被称为原始测试用例，另外一个是由原始测试用例和蜕变关系得到的，被称为衍生测试用例。当并发系统不存在测试预期时，可以利用蜕变测试技术帮助测试软件中的故障。在此之前，首先需要建立面向并发系统的蜕变测试框架。研究如下问题： 2. **并发故障检测的方法：**与传统软件测试不同，即便执行了有潜力揭示软件故障的测试用例，往往还需要在特定的执行流交错下才能揭示并发系统的故障。因此不仅要关注生成测试用例的方式还要注意尽量覆盖可能出现的执行交错，以便增加触发某种交错性质的概率。 3. **并发系统蜕变关系的识别：**在充分了解各种交错模式的基础上，提取不仅包含待测程序属性还包括执行流交错模式特点的蜕变关系。 4. **面向并发系统的蜕变测试框架：**在传统并发系统测试的基础上，考虑蜕变测试的机制与特点，建立面向并发系统的蜕变测试框架。 5. **蜕变关系获取：**蜕变测试技术的核心问题之一是识别蜕变关系。与顺序故障相比，并发故障更难被揭示，原因是即便执行了可能揭示故障的测试用例，也只有在特定的执行交错下才能揭示并发故障。因此与仅考虑并发系统属性的蜕变关系相比兼顾到执行交错属性的蜕变关系可能更好。研究如下问题: 6. **如何提取包含执行交错属性的蜕变关系：**根据并发程序执行的特点，并发故障只有在特定的执行交错下才能被揭示。因此识别的蜕变关系使得衍生测试的执行交错尽量不同于原始测试用力的执行交错。 7. **如何选择识别的蜕变关系：**对于一个确定的并发系统可以识别多个蜕变关系。由于测试资源是有限的，怎么在识别的所有蜕变关系集中选择较少数目的蜕变关系并且到达相同或者相似的测试效率是我们重点研究的问题。 8. **优化面向并发系统的蜕变测试：**在现有的并发系统的测试技术中，超过95%的技术与挑选执行交错有关。然而原始测试用例对揭示并发故障也有重要的作用，但是几乎没有这方面的研究。目前的并发技术的测试用例主要通过两种方式产生，一种是在原有的测试用例集中构建并发测试用例；另一种是随机产生测试用例。随机地产生测试用例虽然是一种简单易用的测试用例生成策略，但是由于该技术没有用到待测程序的信息，因此该方法产生的测试用例不利于揭示并发故障。研究如下问题： 9. **怎么生成高效的测试用例集：**产生更有可能揭示软件故障的测试用例无疑可以提高测试技术的效率，更加软件的可靠性。因此产生高效的测试用例集可以提高面向并发系统的蜕变测试技术的效率。 10. **生成的测试用例集有效性分析：**有很多度量标准可以评价测试技术的有效性，例如F-measure，T-measure，P-measure等。在某种度量指标下以及对应的时间方面比较提出的测试用例生成方式与随机测试的效率 11. **面向并发系统的蜕变测试实例研究及工具支持：**当并发系统缺少测试预期时，我们提出的面向并发系统的蜕变测试技术是否能够有效地揭示并发系统中的故障需要通过实例验证。测试对象对验证我们技术的有效性尤为重要。另一方面，并发系统的测试往往需要分析执行交错，如果仅仅由测试人员进行分析无疑需要消耗大量的资源，因此开发面向并发系统的蜕变测试支持工具可以提高测试效率。研究如下问题: 12. **寻找合适的测试对象：**为了使得我们提出的技术更具有说服力，选择的测试对象的规模要大，如果可能尽量选择开源的并发程序。 13. **开发相应的支持工具：**很多情况下，即便我们能够找到合适的开源并发程序作为测试对象，并发故障的植入依然是需要克服的重大问题。已经有一些产生顺序程序故障的工具例如MuJava、Proteum等，但是这些工具并不支持产生并发故障。人工生成并发故障需要消耗大量的资源，因此开发可以生成并发变异体的工具很有必要。另一方面，开发支持面向并发系统的蜕变测试工具可以进一步地提高测试效率。 14. **技术路线与方法**   本课题将蜕变测试技术运用到并发系统的故障检测中，**解决并发系统缺少测试预期时无法测试的问题，探索并发系统蜕变关系的识别以及优化测试用例的生成方式**。包括面向并发系统的蜕变测试框架、蜕变关系获取、从测试用例生成方面优化面向并发系统的蜕变测试、面向并发系统的蜕变测试实例研究与工具支持。技术路线如下：   1. **将并发系统的执行特点与蜕变测试技术结合构建面向并发系统的蜕变测试框架**。具体来说:(i)并发系统可以根据执行流之间的通信方式分为两类，一种是消息传递类型，另一种是共享内存。无论哪一种类型的并发系统，它们共同完成一个任务时都可能出现不确定的执行交错，并且只有在特定的执行交错下才能揭示软件中的并发故障；(ii)运用蜕变测试技术时，首先需要识别蜕变关系，然后原始测试用例在蜕变关系的作用下生成衍生测试用例，接着分别执行原始与衍生测试用例，并验证结果是否满足蜕变关系。综合这两点提出有效的面向并发系统的蜕变测试框架。 2. **优化面向并发系统的蜕变测试框架**。现有的并发测试技术很少关注测试用例生成这一方面,测试用例要么在已有的测试用例集上重新构建要么随机产生测试用例。然而测试用例对揭示并发故障尤为重要。适应性随机测试在随机测试的额基础上进了改进，很多研究结果表明该技术的测试效率比随机测试技术高。该技术的提出是基于造成故障的输入趋向于集簇在连续的区域这一直觉。该方法的主要思想是：选择多样化的测试用例，使得测试的输入均匀地覆盖整个输入域。   传统的适应性随机测试技术挑选距离执行过的测试用例集“最远”的测试用例。结合并发测试技术的特点：在输入可能揭示故障的测试用例前提下，只有执行特点的交错才能揭示并发故障。因此指导下一个测试用例的挑选规则应当与覆盖未被覆盖的执行交错有关。测试用例覆盖的未被覆盖的执行交错越多越有可能揭示软件中的并发故障。为了使适应性随机测试在并发系统中有更好的表现，我们首先**提出执行交错的覆盖度准则：**在对顺序程序进行测试时，代码的覆盖度准则可以用来评价一个测试用例集的有效性或者判断当前测试是否充分。然而在对并发程序进行测试时，同一个测试用例在同一个并发程序中执行两次可能经历不同的执行过程，从而得到不同的执行结果。随着执行流数目以及软件规模的扩大，执行流之间的交错发生“组合爆炸”。由于测试受时间以及其它资源的限制，我们不可能探索所有的交错可能。我们可以根据常见的交错性质（例如：数据竞争、死锁、原子违反）提出一种覆盖度准则。在该覆盖度准则的指导下，我们不仅能保证软件的质量又能提高测试的效率。然后**提出并发测试用例的多样性度量指标：**经典的适应性随机算法的核心步骤是选择与已经执行测试用例最远的一个测试用例进行测试。为了用更少的测试用例覆盖较多的执行交错，我们可以用测试用例能覆盖尚未被覆盖的交错数目作为测试用例多样性的指标。最后**构建面向并发系统的适应性随机测试框架：**在传统并发系统测试的基础上，考虑适应性随机测试的机制与特点，设计针对并发测试的适应型随机测试算法并建立面向并发系统的适应性随机测试框架。   1. **采用实例研究与变异分析相结合的方式评估面向并发系统的蜕变测试的可行性与有效性**：(i)只有在真正的并发系统上进行验证才能评估并发测试技术的有效性。我们拟采用伯克利大学收集标准程序检测集RADBench为研究对象，RADBench包含了Mozilla SpiderMonkey, Mozilla NSPR, Memcached, Apache Web引擎，Google浏览器并发系统并包含相应的共计10个真实的并发故障。针对对个并发程序实例，在原有的真实并发故障基础上采用变异分析技术产生变异体集合；(ii)针对每一个实例的所有变异体，运用提出的面向并发系统的蜕变测试框架，统计反应测试技术效率的不同度量标准的值，并计算变异得分；(iii)评估面向并发系统的蜕变测试技术的有效性，包括变异得分以及时间开销。 |

**三、拟解决的关键问题与创新之处**

|  |
| --- |
| 1. **拟解决的关键问题** 2. 并发系统的执行方式与传统顺序程序的执行方式有很大的区别，相应地，并发故障与不同于传统顺序程序的故障类型。如何理解并发系统中故障触发，对识别蜕变关系以及并发系统的测试尤为重要。需要解决的关键问题之一是**如何构建面向并发系统的蜕变测试框架**。 3. 识别待测软件的属性是提取蜕变关系的前提。但是对于并发系统而言，除了待测系统本身的属性还需要考虑执行交错的性质。需要解决的关键问题是：**如何获取并发系统的蜕变关系。** 4. 只有执行特定的交错才能揭示并发故障。随着软件规模以及执行流的增加，交错的数量呈指数形式增长。然而测试资源是有限的，探索所有的交错是不可能的。因此在实际的测试过程中需要解决的一个问题是：**如何定义执行交错的覆盖准则。** 5. 与测试顺序程序相比，适应型随机测试不能仅仅考虑测试用例本身还要兼顾到覆盖的执行交错，才能有效地提高并发故障的测试效率。需要解决的关键问题是：**如何构建面向并发系统的适应性随机测试框架**。 6. 为了验证提出技术的有效性，我们需要在实际的程序上进行试验，自动化测试工具可以节省大量的测试资源。我们需要解决的关键问题是：**找到合适的测试程序及开发相应的自动化测试工具**。 7. **本项目的创新之处·**   本课题研究面向并发系统的蜕变测试。预期研究成果将为开发可靠的并发系统提供测  试的理论与工具支持。本项目的创新之处在于：   1. 本课题根据并发系统的运行特点，解决缺少测试预期时的测试问题：当缺少测试预期时，很大一部分并发测试技术将不能对并发系统进行测试。本课题用能够缓解测试预期问题的蜕变测试技术解决缺少测试预期时并发系统无法测试的问题。 2. 引入适应性随机测试技术进一步优化面向并发系统的蜕变测试技术。适应性蜕变测试根据引起故障的输入趋向于集簇在连续的区域这一直觉，旨在均匀的在输入域中选择测试用例。该技术相对于随机测试更有可能选中具有揭示并发故障潜力的测试用例。 3. 构建面向并发系统的蜕变测试框架，开发相应的测试支持工具原型：目前很少的工作涉及到缺少测试预期的并发测试和开发相应的支持工具。由于蜕变测试往往需要提取蜕变关系并且需要分析待测并发程序中的执行交错，提供一个自动化的蜕变关系生成工具以及自动化测试工具，可以提高面向并发系统的蜕变测试揭示故障的效率。 |

**四、与实验室研究工作的关系**（请列出拟开展合作的实验室研究人员）

|  |
| --- |
|  |

**五、合作研究人员意见**（请列出合作基础以及具体的合作计划）

|  |
| --- |
|  |

**六、以往承担实验室开放课题的情况**

|  |
| --- |
| **说明：**请列出在开放课题基金支持下，且署名计算机科学国家重点实验室的，已发表（或已被录用）论文及鉴定或获奖的成果；并提供论文或成果说明的电子版或复印件，也请附上录用函及鉴定或获奖证书的复印件 |
|  |

**七、申请经费及预算**

|  |
| --- |
| **说明：**申请经费主要包括科研业务费[[1]](#footnote-1)〔注一〕、材料费[[2]](#footnote-2)〔注二〕、其它[[3]](#footnote-3)〔注三〕等，预算需合理、详细，“计算依据”需列出费用名称及金额，预算中不得包含固定资产类的设备购置费。 |
| 1. **申请总金额：**元人民币。 2. **总预算：**  |  |  |  | | --- | --- | --- | | **名称** | **金额（元）** | **计算依据（细化预算项目）** | | 科研  业务费 |  |  | | 材料费 |  |  | | 其它 |  |  | | **合计** | **元人民币** | | |
| 其它要说明的问题： |

**八、申请人所在单位学术委员会意见**

|  |
| --- |
| 申请人所在单位学术委员会负责人： 申请人所在单位盖章： |
| 如有推荐人，推荐人意见：    推荐人签字： 推荐人职称：  所在单位： 与申请人关系： |

申请人保证以上所填内容属实并签字： 申请人所在单位领导签字：

申请日期： 年 月 日 单位盖章：

**九、计算机科学国家重点实验室学术委员会审批意见**

|  |
| --- |
| **学术委员会主任签字：**  **年 月 日** |

1. 〔注一〕：科研业务费，指进行课题研究、业务活动所用开支。主要包括机时费、网络费；会议费、差旅费、学术交流费、调研费；报告及论文的印刷费、版面费；图书资料费等。 [↑](#footnote-ref-1)
2. 〔注二〕：材料费，指科研所需材料的购置费。主要包括存储、刻录设备；软件工具；打印、复印材料；其它易耗材料等。 [↑](#footnote-ref-2)
3. 〔注三〕：其它，主要包括协作费、管理费等。协作费指支付给外单位协作者的费用；管理费指项目组织实施所用开支。 [↑](#footnote-ref-3)