|  |
| --- |
| **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，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]。**并发流程之间不确定相互作用与影响的情形通常简称为执行交错**(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。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。由于流程之间执行的次序不确定，每一种流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）**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；（2）**原子违反：**某一流程中应具有原子性的一系列操作与来自其它流程的操作出现交错；（3）**死锁：**同步机制无限期地阻止一些流程继续执行。**由于上述交错性质，并发程序测试在测试用例生成、交错性质分析、测试结果判定等方面面临新的问题与挑战**。  近年来，围绕并发程序的测试用例生成、执行交错分析、测试结果判定问题，人们提出了一系列的并发程序测试技术[1]。部分代表性研究工作包括：   1. **并发程序的测试用例生成方面：**测试用例生成是软件测试的一个重要步骤，直接影响软件测试的故障检测能力。并发程序的测试用例生成技术主要有：（1）随机生成方法[4]：适用于规模较小的并发程序，能够揭示一些简单的并发程序故障类型；（2）顺序测试用例并发化处理方法：从已有的测试用例集中选择那些与并发操作相关的测试用例，然后将这些顺序执行的测试用例构建成并发测试用例。 2. **并发程序执行交错检测方面，**已有技术可以分为**基于故障的交错执行检测技术**和**交错空间探索技术**两大类**。**其中，**基于故障的交错执行检测技术**主要关注待测并发程序的执行交错是否存在某种交错性质，即通过分析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种交错性质的条件；**交错空间探索技术**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。   依据不同的交错性质类型，**基于故障的交错执行检测技术**可以分为**：**   * **面向数据竞争的执行交错检测技术：**首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。并发程序中流程执行某个代码片段或者执行某个指令称为事件，Lamport 等人提出针对并发程序中多个流程中不同事件排序的happens-before 分析方法[5]， 该方法首先对并发程序中的事件进行排序，然后分析排序后的事件是否可能发生数据竞争。由于需要每个执行流程的信息、以及所有的执行流对每个共享数据项的访问信息，该方法存在时间开销大的缺点。为了改进happens-before方法的性能，Flanagan 等人提出了在运行时保存额外信息的动态happens-before方法[6]，该方法仅记录共享数据最近一次写操作，因此减少了分析的对象，有效地降低了时间开销。Marino 等人提出对部分源程序进行采样分析的轻量级探测技术[7]，仅仅分析部分程序而不是整个程序，从而降低了时间开销。基于共享变量经常在没有持锁的情况下被初始化这一观察，Savage 等人提出基于锁集分析的数据竞争检测方法[8]，首先为每一个待测并发程序的共享变量*v*创建一个锁集合*C(v)*，该集合初始时包含所有可能的锁对象；当流程访问*v*时，将*C(v)*与当前流程持有锁形成的集合*T(l)*的交集作为新的*C(v)*。如果*C(v)*为空集，则表明共享变量*v*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，提出一种共享内存程序的数据竞争检测方法[9]，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等人开发了基于锁集分析的并发程序的数据竞争检测工具[10]，该工具可以选择性捕获并发程序中同步操作和内存访问，有助于降低锁集分析技术的时间开销。 * **面向原子违反的执行交错检测技术：**利用执行交错串行化验证程序的正确性。根据具有原子性的区域不同，相关技术可以分为以代码为中心与数据为中心的原子违反检测技术。(1) **以代码为中心的原子违反检测技术：**根据软件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[11]，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，如果最终某个方法（或代码块）的所有事件没有被其它流程的事件交错，则不存在原子违反；否则，检测到原子违反。在Atomizer方法基础上，人们提出了一系列的原子违反检测改进方法：Lu等人提出一种根据待测程序的结构以及执行信息推断原子代码块方法AVIO[12]，有效缓解了无法从软件规格说明书中获得原子代码块的问题；Flanagan等人依据原子代码块的规格说明分析可能引起原子违反的操作，忽略不相关的操作[13]，有助于减少故障误报情形与降低时间开销；(2) **以数据为中心的原子违反检测技术：**该方法首先定义常见的不符合原子集访问的交错场景模式，然后检查在并发执行流程是否存在上述模式。如果存在，则表明存在原子违反故障。原子集表示程序中原子化更新的变量，而定义在原子集上的操作称为工作单元[14]。在原子集与工作单元等概念基础上，Vaziri等人定义了11种原子违反故障类型[15]。类似的，Hammer等人识别了11种可能导致原子违反的数据访问交错场景，提出了相应的自动检测算法[16]。Lai等人利用happens-before分析方法推断Hammer等人识别的可能导致原子违反的数据访问交错场景 [17]。 * **面向死锁的执行交错检测技术：**死锁主要原因是共享资源访问次序不正确，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。Havelund 基于上述观察提出一种运行时分析算法GoodLock [18]，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并构建锁树，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，因此可能存在故障误报的情形；在GoodLock方法基础上，Joshi等人提出了一种面向并发程序的两步骤死锁动态分析方法[19]，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，该方法可以有效降低GoodLock的误报率；为了提高GoodLock方法的性能，Samak 等人提出一种面向锁树的修剪方法[20]、利用顺序执行的程序属性构造死锁检测的并发程序用例生成方法[21]。   依据探索交错空间不同方式，**交错空间探索技术**可以分为**：**   * **并发程序执行交错的压力测试：**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。在并发程序中，压力测试反复执行同一个测试用例集，旨在检查并发程序是否可以线性化或版本变化对并发程序性能的影响。Fonseca等人提出比较输出以及程序内部状态差异检测语义并发故障和潜伏并发故障的方法[22]，首先将同一测试用例的每一次并发执行结果与所有可能的线性化执行结果进行对比，如果并发执行结果和任意一个线性化执行都不相同，并发程序存在语义并发故障。然后利用程序的状态汇总方法捕获每次执行的程序状态，最后利用程序注解比较不同输入下程序状态的差异揭示潜伏的故障。Pradel 等人提出了一种并发程序测试用例生成与执行技术SpeedGun [23]，首先自动产生一组测试用例集，然后在不同版本的并发程序上重复执行测试用例集，并比较不同版本性能的差异。 * **并发程序执行交错穷尽探索：**这方面的研究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，**执行所有可能出现的交错模式需要大量的资源**。一方面，合适的测试用例集可以减小要探索的交错空间。例如，Nistor等人提出了基于随机方法的面向对象并发程序的测试方法Ballerina [24]，首先在测试用例中创建一个待测类的实例和两个随机执行待测对象一个方法的线程，然后利用线性化的概念判断测试执行是否揭示了并发故障。在Ballerina方法的基础上，Pradel等人将并发程序崩溃和死锁作为非线性化执行揭示并发故障的充分条件[25]，降低了传统Ballerina方法误报的故障数目。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等人利用符号执行产生输入数据并开发了相应的支持工具[26]，首先运用符号执行的方式产生测试用例，在执行测试用例的过程中记录程序中的分支节点，然后产生可以覆盖到现有测试用例没有覆盖到的分支。该方法通过在测试的过程中不断产生新的测试用例不断覆盖未被覆盖到的执行交错。 * **并发程序执行交错覆盖标准：**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。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限制要探索的程序代码量。Wang 等人提出了高风险执行交错覆盖标准，首先动态记录共享变量的访问次序，只选择高风险的交错模式（具有较大概率揭示并发故障的执行交错）执行[27]。Hong 等人提出了同步代码区的覆盖标准，在执行过程中动态地构建线程模型，该模型动态捕获并记录线程访问共享内存以及同步代码块的操作，然后执行尚未被覆盖的同步代码块[28]。  1. **并发程序的测试结果判定：**目前多数并发测试技术涉及到两种类型的输出[1]：(1) 待测程序在执行的过程中出现符合某种故障类型（数据竞争、原子违反、死锁等）的执行交错，则认为待测并发程序存在并发故障；(2) 待测并发程序执行测试用例并将输出的结果与预期对比，从而判断待测的并发程序中是否存在并发故障。   **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[29]、断言[30]、机器学习[31]、假设检验[32]、蜕变测试[33]。蜕变测试是一种可以有效缓解测试预期问题的测试技术，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。**围绕蜕变测试理论**、**与其它测试技术结合**、**在不同领域的应用三个维度**，**人们取得了大量的研究成果**[37]**。部分代表性的研究工作包括**：   1. **蜕变测试理论方面**：（1）蜕变测试原始测试用例生成技术：大多数蜕变测试的原始测试用例要么随机生成要么利用已经存在的测试用例作为原始测试用例。接下来展示其它测试用例的生成方式。Gotlieb等人提出了满足特定蜕变关系的原始测试用例和衍生测试用例生成方法[38]，首先将原始程序转化为约束逻辑程序，然后基于预先定义的约束形式构建基于故障的蜕变关系，最后利用约束逻辑程序求解器寻找符合蜕变关系的原始测试用例和衍生测试用例，如果存在这样的测试数据表明程序中存在故障。Chen等人比较特殊值与随机值对蜕变测试效率的影响[39]，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等人提出了最大化覆盖程序路径的遗传算法[40]，旨在产生少量但是高效的原始测试用例。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[41]，首先分析源程序产生符号化的输入，然后人工检查符号化的输入并且提取蜕变关系，最后实例化符号化的输入产生原始测试用例；（2）蜕变关系识别技术**：**Liu等人提出了将多个已存在的蜕变关系复合成新蜕变关系的CMR方法[42]，产生的新蜕变关系应当包含所有原始蜕变关系的属性，并且应具有和原始蜕变关系相似的测试效率。Zhang等人通过反复执行待测程序，从输入与对应的输出之间提取线性或者多项式形式的蜕变关系[43]。Su等人提出了动态识别蜕变关系方法[44]，首先执行待测程序，然后分析程序的执行情况确定识别的蜕变关系与哪些方法相关，最后反馈给测试人员确认。Chen提出了基于范畴划分的蜕变关系识别方法 [45]。补充我们的uMT方法… 2. **蜕变测试与其它测试技术结合**：蜕变测试与其它测试技术相结合，提高其适用性。Dong等人在结构进化测试中引入蜕变测试提高了测试效率[46]，该技术在搜索过程中使用蜕变关系，将原始测试用例和衍生测试案例都视为候选解决方案，加速达到某种覆盖标准；Xie等人在对缺少测试预期的程序利用频谱分析与蜕变测试定位故障[47]，首先执行测试用例并记录测试用例是否违反蜕变关系，然后利用频谱分析的方法定位故障。 3. **蜕变测试在不同领域中的应用：**由于不需要测试预期，蜕变测试在其它领域中也得到了广泛的应用。具体来说：  * **蜕变测试运用到Web服务领域**：Sun等人提出了面向Web服务的蜕变测试框架并开发了半自动化的原型工具[36]，首先从Web服务的WSDL描述文档中提取蜕变关系，然后根据WSDL文档随机产生测试用例并且运用蜕变关系得到相应的衍生测试用例。Chan等人提出了利用蜕变服务封装待测服务的面向SOA的蜕变测试方法学[48][49]，首先为每一个待测服务构建蜕变服务，然后蜕变服务不仅将每一个原始测试用例转发给待测服务也根据识别的蜕变关系生成衍生测试用例并转发给待测服务，最后蜕变服务接受待测服务返回的结果并检验是否违反了蜕变关系。 * **蜕变测试运用到图像处理程序：**Mayer等人利用蜕变测试对几个图像处理程序进行了比较[50][51]。Kuo等人利用蜕变测试对一个解决计算机图像表面可见性问题的算法实现进行测试并揭示了一个真实的故障[52]。 * **蜕变测试运用到嵌入式系统：**Tse等人提出将蜕变测试应用于基于上下文中间件的软件程序[53]。基于上下文的应用程序根据环境中的信息调整其行为，调整行为的过程依赖中间件。他们根据这一原理构建不同的上下文并检查被测程序的结果是否满足某些关系。Chan等人将蜕变测试应用于无线传感器网络中[54]，不仅检查原始测试用例与衍生测试用例的执行结果也检测在执行过程的能源消耗。   课题组在前期研究工作，在蜕变测试、并发程序测试、变异测试等方面研究工作，积累。。。。具体说来。SPE，IJWSR/ICWS/，COMPUTER J。。。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[34][35]。此外课题组开发了面向Web服务的蜕变测试支持工具MT4WS [36]。本项目将基于上述。。。。开展，，，为。。。。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程序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. **面向并发程序的蜕变测试理论、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：**采用多个并发程序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：**采用多个并发程序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。很多并发程序例如：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程序更难测试。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。当并发程序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程序的蜕变测试可以解决这种缺少测试预期的并发程序难测试的问题，保证并发程序的可靠性。  **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 Computer 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 Computer 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. 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. 9. O. Shacham, M. Sagiv, A. Schuster. Scaling Model Checking of Data-races Using Dynamic Information. *Journal of Parallel and Distributed Computing*, Elsevier, 2007, 67(5): 536-550. 10. 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. 11. 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. 12. 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. 13. C. Flanagan, S. N. Freund, J. Yi. Velodrome: a sound and complete dynamic atomicity checker for multithreaded programs. *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI’08)*, ACM Press, 2006, pp. 334-345. 14. L. Wang, S. D. Stoller. Runtime Analysis of Atomicity for Multithreaded Programs. *IEEE Transactions on Software Engineering*, 2006, 32(2): 93-110. 15. 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. 16. C. Hammer, J. Dolby, M. Vaziri. Dynamic Detection of Atomic-Set-Serializability Violations. *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, IEEE Computer Society, 2008, pp. 231-240. 17. 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 Computer Society, 2010, pp. 235-244. 18. 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. 19. 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. 20. 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. 21. 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. 22. P. Fonseca, C. Li, R. Rodrigues. Finding Complex Concurrency Bugs in Large Multi-Threaded Applications. *Proceedings of the 6th European Conference on Computer Systems (EuroSys’11)*, ACM Press, 2011, pp. 215-228. 23. 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. 24. 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 Computer Society, 2014, pp. 727-737. 25. 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. 26. 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. 27. 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. 28. 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. 29. 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. 30. 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. 31. 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. 32. 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. 33. 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. 34. 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. 35. C. 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. 36. 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. 37. S. Segura, G. Fraser, A. B. Sanchez, A. R. Cortes. A survey on metamorphic testing. *IEEE Transactions on software engineering*, 2016, 42(9): 805-824. 38. 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. 39. 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. 40. 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. 41. 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. 42. 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. 43. 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. 44. 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. 45. 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. 46. G. Dong, S. Wu, G. Wang, T. Guo, Y. Huang. Security Assurance with Metamorphic Testing and Genetic Algorithm. *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT’10)*, IEEE Computer Society, 2010, pp. 397-401. 47. X. Xie, W. E. Wong, T. Y. Chen, B. Xu. Metamorphic Slice: An Application in Spectrum-Based Fault Localization. *Information and Software Technology*, 2013, 55(5): 866-879. 48. W. K. Chan, S. C. Cheung, K. R. Leung. Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications. *Proceedings of 5th International Conference on Quality Software (QSIC’05)*, IEEE Computer Society, 2005, pp. 470-476. 49. W. K. Chan, S. C. Cheung, K. R. Leung. A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications. *International Journal of Web Services Research*, 2007, 4(2): 61-72. 50. J. Mayer, R. Guderlei. On Random Testing of Image Processing Applications. *Proceedings of the 6th International Conference on Quality Software (QSIC’06)*, IEEE Computer Society, 2006, pp. 85-92. 51. R. Guderlei, J. Mayer. Towards Automatic Testing of Imaging Software by Means of Random and Metamorphic Testing. *International Journal of Software Engineering and Knowledge Engineering*, 2007, 17(06): 757-781. 52. F. C. Kuo, S. Liu, T. Y. Chen. Testing a Binary Space Partitioning Algorithm with Metamorphic Testing. *Proceedings of the ACM Symposium on Applied Computing (SAC’11)*, ACM Press, 2011, pp. 1482-1489. 53. T. H. Tse, S. S. Yau. Testing Context-Sensitive Middleware-Based Software Applications. *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC’04).* IEEE Computer Society, 2004, pp. 458-466. 54. W. K. Chan, T. Y. Chen, S. C. Cheung, T. H. Tse, Z. Zhang. Towards the Testing of Power-Aware Software Applications for Wireless Sensor Networks. *Proceedings of the 12th Ada-Europe International Conference on Reliable Software Technologies (ICRST’07)*, Springer, 2007, pp. 84-99. |

**二、研究内容、技术路线与方法**

|  |
| --- |
| 1. **研究内容**   在并发程序设计与测试、蜕变测试等领域前沿研究成果基础上，围绕并发程序测试中测试结果判定与高效的测试用例生成等关键问题开展研究，探索面向并发程序的蜕变测试框架与优化技术、研制相应的支持工具原型。主要研究内容如下：   1. **面向并发程序的蜕变测试框架**   软件测试包含测试用例生成、测试执行与测试结果判定主要步骤。测试结果判定的预期问题是软件测试领域尚未有效解决的开放问题。由于并发程序中并发流程执行的不确定性，测试结果判定更加困难，在某些情况下甚至无法确定测试预期。由于现有的并发程序测试技术通常假设已存在测试预期，当并发程序不存在测试预期时，这些技术的有效性则受到了很大的限制。本课题探索无需测试预期的并发程序新型测试方法。蜕变测试利用待测程序存在的某种内在属性（蜕变属性）生成测试用例（包括原始测试用例和衍生测试用例），通过确定具有某种原始测试用例与衍生测试用例对应的输出是否某种关系（蜕变关系）进行测试结果判定。**蜕变测试无须知道某个测试用例的预期输出，因此可以有效缓解测试预期问题。将蜕变测试应用于并发程序时，首先需要建立面向并发程序的蜕变测试框架**。研究如下问题：   * **并发程序的编程模型与故障特点分析：**并发程序设计引入了多线程、消息传递、Fork/Join等同步概念，不同线程访问公共变量时存在约束限制。支持并发的程序设计语言（如C、Java、BPEL等）的并发机制实现方式不同，相应地，并发相关的故障类型与特点也不同。 * **并发程序的故障检测机制:** 测试用例生成是传统软件测试的关键步骤。并发程序测试中，为了增加触发某种交错性质的概率，不仅需要生成具有揭错能力的测试用例，还需要触发特定的执行交错。并发程序的故障检测不仅要关注测试用例生成方式，还要考虑如何尽可能地覆盖不同的执行交错。  1. **面向并发程序的蜕变关系获取方法**   识别蜕变关系是蜕变测试的核心问题。并发程序测试中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检测到并发故障。因此，需要在各种交错模式的基础上，**提取不仅覆盖待测程序内在的蜕变属性、还需要覆盖不同执行交错模式的蜕变关系**。研究如下问题:   * **如何有效识别包含执行交错属性的蜕变关系：**由于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满足衍生测试用例的执行交错尽量不同于原始测试用例的执行交错。 * **如何表示与选择包含执行交错属性蜕变关系：**与常规程序相比，并发程序的蜕变关系需要包含执行交错信息，因此需要解决一个重要问题是如何描述此类新型蜕变关系。对于复杂的并发程序可能存在若干条蜕变关系，在测试资源受限的情形下，如何选择较少数量且不影响故障检测能力的蜕变关系是需要解决的另外一个重要问题。  1. **面向并发程序的蜕变测试用例生成与充分性分析**   原始测试用例与蜕变关系是影响蜕变测试的故障检测能力两个主要因素。在蜕变关系识别与选择的基础上，通过原始测试用例选择进一步提升并发程序的蜕变测试效率。面向并发程序的测试用例中，较为常见的方法是随机测试用例生成。该方法具有简单易用的特点，但没有利用并发程序的特点与执行交错信息，因而并发故障检测能力有待进一步提高。研究如下问题：   * **面向并发程序的高效原始测试用例生成：**增加测试用例集的多样性有助于更有效的检测软件故障。为了提高蜕变测试的故障检测效率，进一步研究面向执行交错的并发程序覆盖率标准，以及生成满足不同覆盖率标准的测试用例方法。 * **面向并发故障的蜕变测试用例集充分性分析：**采用多种度量指标（如F度量、T度量和P度量）评价满足不同覆盖率标准的蜕变测试用例集的故障检测有效性。  1. **面向并发程序的蜕变测试工具研发与评估**  * **面向并发程序的蜕变测试工具原型研制：**并发程序的测试往往需要覆盖数量庞大的执行交错。为了提高面向并发程序的蜕变测试效率，需要开发相应的蜕变测试工具原型，部分支持基于蜕变关系的衍生测试用例生成、基于蜕变关系的测试结果自动判定、生成满足不同覆盖的执行交错场景等。 * **面向并发程序的蜕变测试有效性与性能评估:** 以多个开源并发程序实例为研究对象，采用变异分析方法模拟可能的并发故障，评估面向并发程序的蜕变测试技术的可行性与有效性。  1. **技术路线与方法**   **针对并发程序的不确定性、测试预期问题更加突出的问题，本课题探索将蜕变测试应用于并发程序的关键问题**，包括面向并发程序的蜕变测试框架、面向并发程序的蜕变关系获取、面向并发程序的测试用例生成与充分性分析、面向并发程序的蜕变测试工具原型研制。拟采取的技术路线与方法如下：   1. **通过并发程序的编程模型与蜕变测试原理相结合，构建面向并发程序的蜕变测试框架。**具体来说：（A）**在并发程序的编程模型与故障检测方面**，依据执行交错的通信方式，并发程序分为消息传递类型和共享内存类型。两种类型的并发程序中都存在不确定的执行交错，相应的并发故障类型可以归纳为不同的交错模式，包括数据竞争、原子违反、死锁。针对检测上述并发故障，不仅需要能够执行相应流程的测试用例，并覆盖不同的执行交错；（B）**在面向并发程序的蜕变测试框架时**，首先需要依据被测并发程序的蜕变属性识别蜕变关系，然后进行交错特征分析，在满足交错执行覆盖的基础上进行原始测试用例与衍生测试用例生成，分别执行原始与衍生测试用例，依据定义的蜕变关系验证测试结果。 2. 根据并发故障的特点，**获取并发程序的蜕变关系时,**不仅需要满足待测程序的蜕变属性,还需覆盖不同的执行交错模式。具体来说：（A）**在识别覆盖交错模式的蜕变关系方面，**首先依据规格说明书定位并发相关操作，然后识别并发操作的蜕变属性并提取相应的蜕变关系，依据程序结构在蜕变关系增加执行交错的相关信息，确保原始测试用例或者衍生测试用例的执行符合某种交错性质；（B）**在并发程序的蜕变关系选择方面，**基于使用尽可能少的蜕变关系覆盖尽可能多的符合某种故障类型的执行交错这一思路，首先每一个蜕变关系作用到相同原始测试用例集并得到对应的衍生测试用例集，然后量化每一个衍生测试用例集覆盖符合某种故障类型的执行交错数目，最后根据量化的结果挑选蜕变关系。 3. **在并发程序的原始测试用例生成方面，**通过定义新型交错覆盖准则及依据覆盖准则选择测试用例。具体来说：（A）**定义执行交错的覆盖度准则：**随着线程数量以及程序规模的增加，执行交错数量可能存在“组合爆炸”问题。为了减少覆盖执行交错的数量，依据常见的交错性质（数据竞争、原子违反、死锁）提出相应的覆盖准则，使得测试用例的执行不仅驱动并发流程的执行，同时还覆盖不同的执行交错；**（B）对并发测试用例的充分性度量方面，**采用F度量、T度量和P度量指标评价满足不同覆盖率标准的蜕变测试用例集的故障检测有效性。 4. **采用实例研究与变异分析相结合的方式，评估面向并发程序的蜕变测试的可行性与有效性**：（A）为了模拟更多的并发故障类型，采用变异分析技术对常见并发算法程序集生成变异体集合。通过扩展面向传统Java程序的变异测试工具MuJava，支持并发程序的变异体生成；（B）针对每一个程序实例的变异体集合，运用提出的面向并发程序的蜕变测试技术，统计反映故障检测能力的度量值，计算变异+得分；（C）进一步采用伯克利大学研制的并发程序集RADBench为研究对象（RADBench包含了Mozilla SpiderMonkey、Mozilla NSPR、Memcached、Apache Web引擎、Google浏览器等多种并发程序，并隐藏10个真实的并发故障），评估提出的蜕变测试技术的有效性。 |

**三、拟解决的关键问题与创新之处**

|  |
| --- |
| 1. **拟解决的关键问题** 2. 并发程序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。如何理解并发程序中故障触发，对识别蜕变关系以及并发程序的测试尤为重要。需要解决的关键问题之一是**如何构建面向并发程序的蜕变测试框架**。 3. 识别待测软件的属性是提取蜕变关系的前提。但是对于并发程序而言，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**如何获取并发程序的蜕变关系。** 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**如何构建面向并发程序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程序的蜕变测试技术，预期研究成果将为开发可靠的并发程序提供测试的理论与工具支持。本项目的创新之处在于：   1. 根据并发程序的运行特点，**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程序进行测试。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程序无法测试的问题。 2. **引入适应性随机测试技术进一步优化面向并发程序的蜕变测试技术**。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. **构建面向并发程序的蜕变测试框架**，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，可以提高面向并发程序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)