刘益强

北京科技大学

基于符号执行的蜕变测试中原始测试用例生成技术与支持工具研究

**密　　　　级：\_\_\_\_\_\_\_\_\_\_\_\_\_**

论文题目：基于符号执行的蜕变测试中原始测试用例生成

技术与支持工具研究

S20140931

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

刘益强

作　 　者：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

软件工程

专 业 名 称：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2016年 12月 22日

基于符号执行的蜕变测试中原始测试用例生成技术与支持工具研究

**Research on Symbolic Execution Based Source Inputs Generation Technique in the Context of Metamorphic Testing and Its Supporting Tool**

研究生姓名：刘益强

指导教师姓名：孙昌爱

北京科技大学计算机与通信工程学院

北京100083，中国

Master Degree Candidate： Liu Yiqiang

Supervisor： Sun Chang-ai

School of Computer and Communication Engineering

University of Science and Technology Beijing

30 Xueyuan Road，Haidian District

Beijing 100083，P.R.CHINA

分类号：\_\_\_\_\_\_\_\_\_\_\_\_ 密　　级：\_\_\_\_\_\_\_\_\_\_\_\_\_\_

004.41

TP311

１０００８

ＵＤＣ：\_\_\_\_\_\_\_\_\_\_\_\_ 单位代码：\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**北京科技大学硕士学位论文**

基于符号执行的蜕变测试中原始测试用例

**论文题目：**

生成技术与支持工具研究

刘益强

**作者：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

孙昌爱

北京科技大学

**指 导 教 师： 单位：**

**指导小组成员： 单位：**

**单位：**

**论文提交日期：**2016年 12月 22日

**学位授予单位：北 京 科 技 大 学**

致 谢

两年半的研究生学习生活转眼间就要结束了，在此期间的学习与生活中我得到了孙昌爱导师、同学们及亲友们的鼎力帮助与支持。值此论文完成之际，向他们对我提供的无私帮助表示真诚的感谢。

首先，本论文能顺利的完成，很大程度上得益于我的导师孙昌爱教授，在本课题的选题及研究过程中得到了孙老师的悉心指导。孙老师多次询问课题的研究进程，并为我指点研究思路与方法，对于研究中的难题细心点拨。孙老师严谨求实的治学精神与科学态度，一丝不苟的工作作风，踏踏实实的工作精神，不仅传授我科学知识，而且还教会我了做人的道理，这些使我终生受益。在此谨向孙老师表示诚挚的谢意和崇高的敬意。

其次，感谢520实验室的所有同学，感谢他们在学习上、生活上给我的热心帮助与鼓励支持，我们一起度过了两年多的快乐时光，使我终生难忘；感谢舍友在生活上对我热情的关心，还有计通研147班全体同学对我生活学习上的帮助。

最后，感谢计算机与通信工程学院的所有老师，感谢他们为我们打下坚实的专业知识基础；同时还要感谢我的父母，在我研究生生活中给予我的支持和鼓励，还要特别地感谢此次参加论文评审和答辩的各位老师们。

正因为有了以上关心我照顾我的人，我的研究生生活才能够圆满的结束，此次毕业论文才会顺利完成。在此，我衷心地感谢他们！

摘 要

软件测试是一种广泛使用的软件质量保证手段。大多数的软件测试方法假设测试预期存在，然而在很多情况下很难甚至不可能获取程序的预期输出，这种情况称为测试预期问题。蜕变测试利用被测程序的蜕变关系生成测试用例，通过验证原始测试用例和衍生测试用例的输出结果是否满足蜕变关系判定程序的测试结果，因此可在测试预期不存在的情况下对程序进行测试，有效地缓解测试预期问题。前期研究工作发现，原始测试用例严重影响蜕变测试的故障检测能力。

本文从原始测试用例的生成与选择角度，探索如何进一步改进蜕变测试的故障检测能力，提出一种基于符号执行的原始测试用例生成技术和一种基于路径距离的原始测试用例优先级排序方法，开发相应的支持工具。本文取得的成果如下：

* **提出了一种基于符号执行的原始测试用例生成技术**：针对源程序进行符号执行，获得程序所有路径约束条件，通过约束求解技术生成满足路径约束的原始测试用例。
* **提出一种基于路径距离的原始测试用例优先级排序方法**：依据测试用例覆盖的程序路径距离，对原始测试用例进行优先级排序。在测试资源有限的情况下，该方法通过调整测试用例的执行次序进一步提高蜕变测试的故障检测能力。
* **开发了基于符号执行的原始测试用例生成技术的支持工具**：该工具可以自动化地生成面向路径覆盖的原始测试用例，并已经集成到课题组前期工作开发的蜕变测试工具MT4WS中，进一步提高了蜕变测试的自动化程度。
* **采用实例程序评估了所提原始测试用例生成技术可行性与原始测试用例优先级排序方法的有效性**：采用实例研究的方式，验证了基于符号执行的原始测试用例生成技术可行性与有效性；评估了基于路径距离的原始测试用例优先级排序方法的有效性。实验结果表明所提方法能有效的提升蜕变测试的故障检测能力。

本文从路径覆盖角度探索原始测试用例生成问题，结合符号执行和约束求解的思想，提出一种高效的原始测试用例生成技术和一种基于路径距离的原始测试用例优先级排序方法并开发了支持工具，提高了蜕变测试的故障检测能力，同时提高了蜕变测试的自动化程度。

关键词：蜕变测试，原始测试用例，符号执行，约束求解

Research on Symbolic Execution Based Source Inputs Generation Technique in the Context of Metamorphic Testing and Its Supporting Tool

Abstract

Software testing is a widely used software quality assurance method. Most software testing techniques assume the existence of the test oracle when they are used. However, in many practical situations, it is difficult or even impossible to obtain the expected outputs of the program, which is known as the oracle problem. Metamorphic testing was proposed to alleviate the oracle problem, which leverage the metamorphic relations of a program under test to generate test cases, and verify the outputs of source test cases and follow-up test cases, thus is able to test the program without a need of oracles. In our previous work we have observed that source test cases play a key role in fault detection effectiveness of metamorphic testing.

In this thesis, we explore the improvement of fault detection effectiveness of metamorphic testing from the perspective of source test cases generation and prioritization. We proposed a symbolic execution based source test case generation technique, and a path distance based test cases prioritization technique, and developed a supporting tool. The main contributions made in this thesis are as follows:

* **A symbolic execution based source test case generation technique**: Obtain all path constraints of the program according to the symbolic execution results, for each path constraint, the source test case which satisfies it is generated using the constraint solving technique.
* **A path distance based test cases prioritization technique:** According to the program path distance covered by the source test case, the priority of the source test case is determined. This technique further improves the fault detection effectiveness of the test case by adjusting the execution order of them.
* **A supporting tool for the symbolic execution based source test case generation technique:** A tool for path-oriented automated source test case generation is designed and actualized, which has been integrated into the pre-work tool for metamorphic testing (MT4WS) to improve the automation.
* **Effectiveness evaluation of the source test case generation and prioritization technique:** Three case studies were conducted to verify the feasibility and effectiveness of the symbolic execution based source test case generation technique, and the effectiveness of a path distance based test cases prioritization technique. Experimental results show that the methods we proposed can effectively improve the fault detecting ability of metamorphic testing.

In this thesis, we explore the problem of source test case generation by a path covering perspective. Under the combination of symbolic execution and constraint solving, we proposed a source test case generation technique and developed a supporting tool, which improved the fault detection effectiveness and the automation of metamorphic testing.

Key Words： Metamorphic Testing, Source Test Case, Symbolic Execution, Constraint Solving

目 录

[致 谢 I](#_Toc470004729)

[摘 要 III](#_Toc470004730)

[Abstract V](#_Toc470004731)

[插图和附表清单 IX](#_Toc470004732)

[1 引言 1](#_Toc470004733)

[2 背景介绍 2](#_Toc470004734)

[2.1 相关技术及工具 2](#_Toc470004735)

[2.1.1 蜕变测试 2](#_Toc470004736)

[2.1.2 模型检测及工具JPF 4](#_Toc470004737)

[2.1.3 符号执行及工具SPF 6](#_Toc470004738)

[2.1.4 约束求解工具choco 8](#_Toc470004739)

[2.2 国内外研究现状 10](#_Toc470004740)

[2.2.1 蜕变测试研究现状 10](#_Toc470004741)

[2.2.2 符号执行研究现状 12](#_Toc470004742)

[2.3 研究背景与意义 13](#_Toc470004743)

[2.4 研究内容与成果 14](#_Toc470004744)

[2.5 论文组织结构 14](#_Toc470004745)

[3 基于符号执行的原始测试用例生成技术 16](#_Toc470004746)

[3.1 基本思想 16](#_Toc470004747)

[3.2 相关定义及概念 16](#_Toc470004748)

[3.3 原始测试用例生成 17](#_Toc470004749)

[3.4 原始测试用例优先级排序 19](#_Toc470004750)

[3.5 方法示例 20](#_Toc470004751)

[4 基于符号执行的原始测试用例生成工具的设计与实现 24](#_Toc470004752)

[4.1 需求分析 24](#_Toc470004753)

[4.2 系统设计 26](#_Toc470004754)

[4.3 工具详细设计 28](#_Toc470004755)

[4.4 系统演示 29](#_Toc470004756)

[4.5 本章小结 34](#_Toc470004757)

[5 实验评估 35](#_Toc470004758)

[5.1 研究问题 35](#_Toc470004759)

[5.2 实验步骤 35](#_Toc470004760)

[5.3 实验对象 36](#_Toc470004761)

[5.4 实验结果分析 46](#_Toc470004762)

[5.5 本章小结 55](#_Toc470004763)

[6 工作总结与展望 56](#_Toc470004764)

[参考文献 57](#_Toc470004765)

[作者简历及在学研究成果 63](#_Toc470004766)

[独创性说明 65](#_Toc470004767)

[关于论文使用授权的说明 65](#_Toc470004768)

[学位论文数据集 1](#_Toc470004769)

插图和附表清单

[图 2-1 蜕变测试过程 3](#_Toc470004770)

[图 2-2 模型检测过程 4](#_Toc470004771)

[图 2-3 JPF顶层架构图 6](#_Toc470004772)

[图 2-4 一个简单程序和对应的符号执行树 7](#_Toc470004773)

[图 2-5 SPF顶层架构图 8](#_Toc470004774)

[图 2-6 SPF配置文件示例 8](#_Toc470004775)

[图 2-7 choco框架 9](#_Toc470004776)

[图 2-8 choco示例程序 9](#_Toc470004777)

[图 3-1 示例程序 16](#_Toc470004778)

[图 3-2 原始测试用例生成原理 17](#_Toc470004779)

[图 3-3 基于路径距离的原始测试用例优先级排序算法 19](#_Toc470004780)

[图 3-4 多分支示例程序 20](#_Toc470004781)

[图 3-5 多分支示例程序流程图 21](#_Toc470004782)

[图 4-1 MT4WS中原始测试用例生成模块用例图 24](#_Toc470004783)

[图 4-2 MT4WS系统框架图 26](#_Toc470004784)

[图 4-3 测试用例自动生成器结构图 28](#_Toc470004785)

[图 4-4 路径信息存储结构 28](#_Toc470004786)

[图 4-5 MT4WS欢迎界面 29](#_Toc470004787)

[图 4-6 Operation选择界面 30](#_Toc470004788)

[图 4-7 蜕变关系定义界面 30](#_Toc470004789)

[图 4-8 原始测试用例生成界面 31](#_Toc470004790)

[图 4-9 测试配置界面 31](#_Toc470004791)

[图 4-10 蜕变关系集管理界面 32](#_Toc470004792)

[图 4-11 测试用例配置界面 32](#_Toc470004793)

[图 4-12 测试执行界面 33](#_Toc470004794)

[图 4-13 测试报告界面 33](#_Toc470004795)

[图 5-1 航空行李托运计费程序:原始测试用例选择对比实验结果 54](#_Toc470004796)

[图 5-2 联通计费程序:原始测试用例选择对比实验结果 54](#_Toc470004797)

[图 5-3 停车计费程序:原始测试用例选择对比实验结果 54](#_Toc470004798)

[表 3-1 多分支示例程序路径信息 22](#_Toc470004799)

[表 3-2 测试用例生成结果 22](#_Toc470004800)

[表 3-3 测试用例优先级排序结果 23](#_Toc470004801)

[表 5-1 航空行李托运计费程序输入参数描述 37](#_Toc470004802)

[表 5-2 航空行李托运计费程序benchmark设置 37](#_Toc470004803)

[表 5-3 航空行李托运计费程序蜕变关系集 38](#_Toc470004804)

[表 5-4 中国联通计费程序输入规格说明 40](#_Toc470004805)

[表 5-5 中国联通计费程序计费标准说明 41](#_Toc470004806)

[表 5-6 中国联通计费程序蜕变关系描述 42](#_Toc470004807)

[表 5-7 停车计费程序输入参数描述 44](#_Toc470004808)

[表 5-8 停车计费单价计算规则 45](#_Toc470004809)

[表 5-9 停车计费程序蜕变关系集 46](#_Toc470004810)

[表 5-10 航空行李托运计费程序变异得分 49](#_Toc470004811)

[表 5-11 航空行李托运程序对比实验结果 50](#_Toc470004812)

[表 5-12 联通计费程序实验结果 51](#_Toc470004813)

[表 5-13 联通计费程序对比实验结果 52](#_Toc470004814)

[表 5-14 停车计费程序实验结果 53](#_Toc470004815)

[表 5-15 停车计费程序对比实验结果 53](#_Toc470004816)

1. 引言

软件测试是软件工程中保证软件质量的一种重要方法，贯穿于软件开发的整个周期。测试人员通过设计并执行测试用例，比较测试用例实际输出与预期输出是否一致来判定程序是否潜藏故障。然而在实际测试中，有可能构造预期输出需要花费相当大的测试资源，甚至有可能测试人员无法判断所构造的预期输出是否正确。这类情况在软件测试中被称为测试预期问题[1]。测试预期问题限制了许多测试方法的实际应用[2,3]。

为了缓测试预期问题，Chen等人[4, 5]于1998年提出了蜕变测试。蜕变测试的核心思想是通过执行一组测试用例，比较对应输出是否满足程序的固有属性来测试程序，这些固有属性被称为蜕变关系。蜕变关系描述了多个测试用例之间输入输出满足的关系。在蜕变测试过程中，不需要构造每一个测试用例的预期输出，因此有效的缓解了测试预期问题。

目前蜕变测试主要研究工作侧重如何通过获取与选择蜕变关系来改进蜕变测试的故障检测能力。在我们的前期研究工作中发现，原始测试用例也严重影响蜕变测试的故障检测能力。生成高效的原始测试用例能有效提高蜕变测试的故障检测能力，然而这方面的研究工作较少。

课题组前期工作针对SOA环境下的Web服务测试问题，提出了一个面向Web服务的蜕变测试框架以及蜕变关系描述语言MRDL，并且基于此框架开发了面向Web服务的蜕变测试工具MTWS[6]，该工具能在不需要构造预期输出的情况下，有效的对Web服务进行测试。经过不断的改进，目前工具除了支持多种蜕变关系定义方式、多种原始测试用例生成方式之外，还支持基于数据变异的蜕变关系获取技术、迭代蜕变测试以及复合蜕变关系的构造。

本文从原始测试用例生成与优先级角度，进一步研究如何改进蜕变测试的故障检测能力。结合符号执行和约束求解技术，提出一种原始测试用例自动生成方法，该方法能生成高覆盖率的原始测试用例。另外，根据路径距离，提出一种原始测试用例优先级排序方法。开发了相应的支持工具，提高了蜕变测试的自动化程度。

1. 背景介绍

本章介绍本文涉及到的相关技术、国内外研究现状、研究的背景与意义以及论文的组织结构。

* 1. 相关技术及工具

介绍论文中涉及的相关技术及工具，包括蜕变测试、模型检测、符号执行和约束求解。

* + 1. 蜕变测试

传统的软件测试技术需要确定测试用例的预期输出，通过比较实际输出结果和预期输出是否一致来判定是否潜藏故障。然而在现实中，获得程序的预期输出有时候需要花费相当大的测试资源，甚至在一些情况测试人员无法获得程序的预期输出，这给传统的软件测试带来了巨大的挑战。

蜕变测试[4, 5]是一种有效缓解测试预期问题的软件测试技术。蜕变测试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以及对应的输出之间应该满足某种关系，这种关系来自于程序固有属性，Chen将这种关系称为蜕变关系。每一组的测试用例包含两个对应的测试用例，原先设计好的测试用例称为原始测试用例（原始测试用例可以通过其他的测试用例生成技术获得），由原始测试用例和构造好的蜕变关系得到的测试用例称为衍生测试用例。

蜕变测试的形式化定义如下[7]：

假设程序*P*是待测函数*f*的实现，*x1, x2, …, xn (n>1)*是函数的不同变量，如果存在输入之间存在关系*R*使得*f(x1), f(x2), …, f(xn)*满足关系*Rf*，即：



则*MR=(R, Rf)*为程序*P*的蜕变关系，显然如果程序*P*正确，则有：



成立，其中*I1, I2, …, In*是对应于*x1, x2, …, xn (n>1)*的测试用例。

蜕变测试过程如图 2-1，首先根据待测程序的规格说明设计蜕变关系并设计原始测试用例，根据原始测试用例和蜕变关系可以生成衍生测试用例。进行蜕变测试时，需要分别执行原始测试用例和衍生测试用例，通过比较两者输出是否满足蜕变关系来判定程序是否潜藏故障。蜕变测试只需要判断多次执行的结果之间是否满足蜕变关系，而不需要得到每个测试用例的预期输出，因此蜕变测试能有效的缓解测试预期问题。



图 2-1 蜕变测试过程

蜕变测试的关键问题有三点：

1. **生成故障检测能力高的原始测试用例**

在蜕变测试过程中，测试用例的生成作为重要的一个环节，衍生测试用例可以通过原始测试用例和蜕变关系中的输入关系生成，因此原始测试用例的生成显得至关重要。在实际测试过程中，测试人员可以综合不同的方法生成原始测试用例，以提高蜕变测试的故障检测能力[8-11]。另外还可以借助构造的蜕变关系，根据已有的原始测试用例迭代生成原始测试用例集，因为蜕变关系包含程序的属性，因此迭代出来的原始测试用例故障检测能力会比随机生成的原始测试用例高[12,13]。

1. **设计高效的蜕变关系**

蜕变关系作为蜕变测试的核心，直接影响蜕变测试的故障检测能力。对于不同类型的错误来说，不同的蜕变关系有不同的检错能力,如何选择蜕变关系是蜕变测试中一个重要的问题。目前，蜕变关系的获取主要靠测试人员根据程序规格说明设计。在不断的经验研究中，已有一些基本的设计准则指导测试人员构造蜕变关系[14-17]。另外，Liu等人提出了一种复合蜕变关系获取方法[18]，通过已有蜕变关系集复合生成新的蜕变关系，从而提高了蜕变关系的获取效率。

1. **提高蜕变测试的自动化程度**

蜕变测试的自动化程度依赖于自动化获取原始测试用例和自动化构造蜕变关系。近几年来，学术界在蜕变测试自动化的研究上取得了一定的成果[19-22]。在前期工作中，课题组提出了一个面向Web服务的蜕变测试框架[6, 23]，并开发了面向Web服务的蜕变测试工具MT4WS。另外，该工具还集成基于数据变异的蜕变关系获取技术[24]，进一步提高了蜕变测试的自动化程度。

* + 1. 模型检测及工具JPF

模型检测[25]是一种有效的形式化验证方法，相关的研究开始于上世纪八十年代，Clarke提出了一种用来描述并发系统的CTL逻辑，并设计了一种算法来检测有穷状态系统是不是满足CTL逻辑公式，为并发系统的自动化验证开辟了一个新方向。

模型检测的基本思想是用状态迁移系统(S)表示系统的行为，用模态/时序公式(F)表示系统的属性。系统是否满足期望的性质就转换为数学问题“状态迁移系统S是否满足公式F”。简单来说，模型检测是一套用于判断软硬件设计的理论模型是否满足规范的方法。对于一个给定的有穷状态并发系统和表示系统性质的表达式，然后利用穷举搜索算法来判定系统是否满足给定的规范。模型检测过程如所示:

系统模型

系统性质

模型检测器

检测结果

是否通过

是

否

性质满足

反例

图 2-2 模型检测过程

模型检测能够完全自动的对系统进行检测，该方法的成功得益于有效的软件测试工具的支持。目前常见的模型检测工具有SMV、SPIN、VeriSoft、SLAM以及JPF等。

JPF (Java Pathfinder)是美国NASA开发的基于java程序的模型检测工具[27]。从本质上来说，JPF就是一个特殊的java虚拟机，是一种显式状态的软件模型检测工具，它与普通的java虚拟机的最大区别在于，普通的虚拟机一次只能执行一条程序路径，而JPF从理论上来说可以执行程序中可能出现的所有路径，即能够执行程序多次，JPF沿着这些路径能够检测出程序中是否存在死锁、未处理异常等违反性质的冲突。如果发现错误，JPF将报告导致程序错误的整个执行路径。不同于正常的调试，JPF跟踪达到错误的每一步，在执行过程中能够进行状态存储、状态匹配和回溯。

JPF是一个可扩展的框架。是JPF的顶层架构。JPF的核心是对虚拟机和搜索策略的设计。虚拟机是状态生成器，搜索策略支持回溯、状态匹配和非确定性调度进行决策。当程序运行在这个特殊的虚拟机上的时候，JPF构造出程序的状态空间，而状态空间的转换时一个线程执行的字节码指令序列，序列的第一条指令代表对应线程上下文转换中一个不确定的选择，在每一个转换边界，JPF保存当前运行的程序状态，目的是为状态回溯和状态匹配。

除了虚拟机和搜索策略，JPF还包括以下的模块：

1. **选择生成器**：JPF使用一个通用的状态空间模型，包括选择器和转换器，并且沿着一条特定的程序执行路径恢复原状映射，这些映射包括程序的堆、当前程序运行的位置、当前运行线程的状态（执行、等待、死锁等）和相应线程操作数堆栈。选择器识别程序中可能的执行状态，如选择哪一个线程作为下一个要执行的线程。转换器中是两个状态之间执行的指令序列，从一个特定的选择值开始，到下一个选择操作结束。JPF在选择生成器中捕获状态空间的分支节点，以便达到跟踪回溯选择状态集的目的。
2. **字节码工厂**：标准的Java虚拟机为程序操作定义了一组字节码指令，如方法的调用、访问字段和操作数进栈出栈操作。JPF中用一个专用的指令来代替每一个字节码，它提供了一个特定的execute( )方法定义了各自的执行语义，JPF内部存储的每一个方法都与代码相关联，因此将其作为一个指令类的数组来处理。
3. **模型java接口**：类比标准的Java本地接口（JNI），与JPF相关的机制是模型Java接口（MJI），MJI允许监听所有的方法，包括构造函数和静态初始化，因此MJI是一个库抽象机制。使用MJI是为了能够在JPF中运行本地Java方法，如果不使用MJI，Java标准库的一些应用程序将无法执行，这些应用程序通常会用到大量的本地Java方法。
4. **监听器**：监听器是JPF一个重要扩展机制，它能够监听控制JPF的内部操作。JPF已经设计了一系列的监听器监听程序运行状态，同时JPF提供了Listener接口供使用者去实现自定义的监听器。
5. **报告系统**：模型检测不仅仅是为了发现程序中存在的故障，更重要的是告诉用户导致漏洞的整个执行过程，目的是为了方便用户对发现的故障进行修复。JPF以报告系统的形式，把系统验证的结果显示给用户。



图 2-3 JPF顶层架构图

* + 1. 符号执行及工具SPF

符号执行[28, 29]的研究开始于上世纪七十年代，但直到近年来才在实践中得到广泛的应用，主要得益于计算机硬件资源的发展、搜索算法的改进以及约束求解技术的推进。

符号执行的核心思想是用符号值表示程序的输入变量，用符号表达式表示程序中的变量和输出结果，通过模拟程序的执行进行程序分析和验证。符号执行的每一步的状态包括三个部分：

1. **路径条件PC（Path Condition）**：路径条件又称为路径约束，是由一系列的分支条件构成。路径条件刻画了一条程序的执行路径，所有满足该路径条件的输入执行的都是同一条程序路径。
2. **程序计数器**：定义了下一条要执行的语句。
3. **程序变量的符号值**：程序中的变量由一个符号表达式来表示。

将符号执行的每一步的状态结合起来就构成了一棵符号执行树，符号执行树表示程序所有的执行路径，程序状态用节点表示，状态的迁移用弧线表示。给出了一个简单的程序和对应的符号执行树。程序开始时路径条件置为true，在程序执行过程中，用符号值X和Y分别替代变量x和y，当执行到判断语句路径一分为二，分别代表条件满足和条件不满足时的执行路径，从而路径条件变成X<Y和X>=Y。这样得到程序可执行的两条路径以及路径的约束条件。



图 2-4 一个简单程序和对应的符号执行树

符号执行工具SPF(Symbolic PathFinder)是JPF的一个扩展，结合了模型检测和符号执行。SPF中程序用符号输入值代替具体的输入值，程序的变量用符号表达式表示，通过分析不同的执行路径生成对应的路径约束条件，这些约束条件可以用约束求解器进行求解，从而生成满足对应执行路径的测试用例。

显示了SPF的顶层结构。SPF依赖于模型检测工具JPF，在JPF提供的虚拟机上能够系统的探索程序中不同的符号执行路径，另外SPF可以使用JPF内置的策略对状态空间进行搜索，如深度优先（DFS）或广度优先（BFS）。SPF采用限制搜索深度的方法来避免程序中存在递归或者循环时程序状态空间爆炸的情况。符号执行过程得到符号执行树中的路径约束条件使用约束求解器进行求解，以此来确定路径是否可执行以及生成相应路径上的测试用例。

为了方便程序的扩展，SPF采用配置文件的方式对符号执行过程的信息进行配置，包括源文件和字节码文件的位置等基本信息。另外需要注意的是，为了使一个方法进行符号执行，用户需要指定方法的哪些参数是符号值，哪些参数是具体值。给出一个SPF的配置示例。



图 2-5 SPF顶层架构图



图 2-6 SPF配置文件示例

* + 1. 约束求解工具choco

choco[30]是一个实用的约束求解工具，提供了问题建模和约束求解的Java接口，并将问题建模和求解过程进行了明确的分离,更加便于用户使用。choco实现了70多种约束其中包含经典的算术约束，reified约束，表约束，全局约束等，同时能实现最流行的tree和geost约束。显示了choco的结构。



图 2-7 choco框架

choco由两部分构成：（1）问题建模，管理问题模型中所涉及到的变量之间的约束关系，并向用户提供了API，使得使用更加方便；（2）问题求解，求解包含内存管理和树搜索。

图 2-8是一个简单的choco使用示例。首先需要创建一个模型对象用于对问题进行建模，然后创建变量并将变量之间的约束关系添加到问题模型中，调用求解器的solve方法对相应的约束条件进行求解，最终获得满足约束条件的可行解。

|  |  |
| --- | --- |
| 1 | **import** org**.**chocosolver**.**solver**.**Model**;** |
| 2 | **import** org**.**chocosolver**.**solver**.**variables**.**IntVar**;** |
| 3 | public class Example **{** |
| 4 | public static void main**(**String**[]** args**)** **{** |
| 5 | // 1. 创建一个模型 |
| 6 | Model model **=** **new** Model**(**"my first problem"**);** |
| 7 | // 2. 创建变量 |
| 8 | IntVar x **=** model**.**intVar**(**"X"**,** 0**,** 5**);** |
| 9 | IntVar y **=** model**.**intVar**(**"Y"**,** **new** int**[]** **{** 2**,** 3**,** 8 **});** |
| 10  11 | // 3. 加入约束  model**.**arithm**(**x**,** "+"**,** y**,** "<"**,** 5**).**post**( );** // x + y < 5 |
| 12 | model**.**times**(**x**,** y**,** 4**).**post**( );** // x \* y = 4 |
| 13 | // 4. 问题求解 |
| 14 | model**.**getSolver**( ).**solve**( );** |
| 15 | // 5. 打印出结果 |
| 16 | System**.**out**.**println**(**x**);** // X = 2 |
| 17 | System**.**out**.**println**(**y**);** // Y = 2 |
| 18 | **}** |
| 19 | **}** |

图 2-8 choco示例程序

* 1. 国内外研究现状

本节介绍蜕变测试与符号执行国内外研究现状与进展。

* + 1. 蜕变测试研究现状

蜕变测试是一种有效缓解测试预期问题的软件测试技术，为软件测试研究开辟了一个新方向。蜕变测试自提出后引起了国内外学者的广泛研究。本文将从以下几方面介绍：

1. **蜕变测试在不同领域的应用**

Xie等人[31]将蜕变测试技术应用于测试与验证机器学习分类算法实现软件的正确性，并通过实例研究验证蜕变测试技术在预期不存在的情况下测试程序的有效性；Segura等人[32]应用蜕变测试技术为特征模型分析工具自动生成测试用例从而避免测试预期问题；Murphy等人[33]将蜕变测试技术应用在医疗领域，并通过实例研究验证了蜕变测试在医疗仿真软件上突出的测试效果；Poon等人[34]针对测试电子表格程序时的问题，提出了将蜕变测试技术应用于测试此类程序，实验研究发现蜕变测试在测试此类程序时表现出很高的故障检测能力。

路晓丽等人[35]将蜕变测试方法用于面向服务软件的单元测试和集成测试过程。姚奕等人[36]利用蜕变测试技术发现由于整型错误产生的隐式错误，实验结果显示基于蜕变关系的整型故障检测方法能够检测出平时发现不了的隐式非预期输出。黄松等人[37]提出将蜕变测试应用于坡度、坡向量算程序的测试中，通过分析坡度、坡向量算程序功能的几何属性、数值计算特性以及具体实现算法等，从中提出蜕变关系进行蜕变测试。林仁超等人[38]针对GIS空间分析程序特点，提出了一套适用于空间分析程序的蜕变测试框架，并且给出了每个组成部分的实现方法。张晶等人[39]基于蜕变测试技术，提出了数据挖掘领域的软件评测方法。

**2) 蜕变测试技术与其他技术结合**

Murphy等人[40]将Java建模语言(JML)和执行断言检查与蜕变测试技术结合，成功测试了存在预期问题的应用程序；Xie等人[41]提出了蜕变切片的概念，将蜕变测试技术基于谱的错误定位技术结合，很好的避免了原有错误定位技术中存在的测试预期问题；Liu等人[42]将蜕变测试技术与一些容错技术相结合，扩充了这些容错技术在预期不存在的情况下的可用性。

**3) 蜕变测试的理论研究**

Asrafi等人[16]通过实例研究得出了具有低程序覆盖的蜕变关系的测试效果不足，但是具有高覆盖的蜕变关系却不一定就有较高的检测效果的结论。Liu等人[18]提出了一种组合蜕变关系构造方法，该方法是在原有蜕变关系的基础上通过将两个旧蜕变关系组合为一个新的更加具有复杂条件的蜕变关系，实验得出组合蜕变关系的故障检测效率很高。董国伟等人[43]在命题逻辑的推理规则的基础上，提出了复合蜕变关系的构造方法，该方法具有更强的检错能力。王瑢等人[17]通过对典型程序测试的案例对蜕变关系的构造进行分析，归纳总结了若干条基本准则用于指导构造蜕变关系。林仁超等人[44]提出了基于复合函数的蜕变关系构造方法，并通过实验验证了该方法对于形式化表示的蜕变关系具有较强适用性和可操作性。Hui等人[45]提出了蜕变关系的通用定义模型，以便重用已有的蜕变关系描述，他们通过构建蜕变关系的分解模型并给出了三个子关系来降低蜕变关系描述的复杂性。Kanewala等人[46]提出了一个通用的方法，应用机器学习相关方法将样本蜕变关系作为训练集，预测生成相似的蜕变关系，并且在科学计算程序中进行了实验研究，得到了很好的蜕变关系相似集。

1. **本课题组前期研究**

课题组前期工作中，提出了面向Web服务的蜕变测试框架[6, 23]，依据该框架实现了面向Web服务的蜕变测试工具MT4WS，经过实验室课题组成员对工具的不断改进，工具支持蜕变关系定义、基于蜕变关系的测试用例生成、测试执行、基于蜕变关系的测试结果判定、测试配置、生成测试日志及测试报告等基本功能。

课题组前期对迭代蜕变测试技术进行改进，提出了一个面向Web服务的迭代蜕变测试技术[47]，该技术首先需要应用其他测试用例生成技术来构造一个原始测试用例集，使用多条蜕变关系依此产生衍生测试用例，若该衍生测试用例和原始测试用例集中的任何一个测试用例不重复，且符合输入的规范，则将该衍生测试用例添加到原始测试用例集中，若生成的衍生测试用例和原始测试测试用例中的测试用例重复或者不符合输入规范，则舍弃该衍生测试用例，循环多次后来获得最终的原始测试用例集。面向Web服务的迭代蜕变测试技术能够从时间和规模上大大提高Web服务测试用例的生成效率，且应用该方法所得到的测试用例集的故障检测能力优于随机生成的测试用例。另外，课题组设计并实现了迭代蜕变测试方法的支持工具，相应的功能模块已经集成到MT4WS中，并对该方法进行了进一步实例研究。

针对蜕变关系的获取问题，课题组前期工作中提出了一种基于数据变异的蜕变关系获取技术[24]。该技术结合蜕变测试和数据变异原理，总结了7类数值型变异算子以及对应的蜕变关系映射规则，提高了蜕变关系获取的效率，同时提高了蜕变测试的自动化程度。目前基于数据变异的蜕变关系获取技术以作为一个功能模块集成到MT4WS。

* + 1. 符号执行研究现状

符号执行这一概念最早在1976年由James C. King博士提出[28]，但受制于当时的计算能力、相应的软件理论和约束求解器的性能等因素，符号的研究直到近十年才流行起来。下面介绍符号执行的主要难点和研究现状。

**1) 路径爆炸问题**

路径爆炸问题指程序的路径数随程序中的分支语句的数量呈指数级增长。路径爆炸问题是符号执行中的主要难点，当前的研究主要从四个方面缓解路径爆炸问题。

1. **基于摘要的方法**

基于摘要的方法的核心思想是通过避免执行程序中的特定代码块来缓解路径爆炸问题。具体的实现方法之一就是将一些重复执行的代码块，如函数，在第一次运行时生成摘要，在下次执行代码块的时候直接使用该摘要，从而避免重复执行该代码块，该方法有效缓解了代码块因调用次数增多和调用深度增加引起的路径爆炸问题。Anand等人利用过程内分析的方法对函数生成摘要来辅助符号执行的过程间分析，有效的缓解因调用函数过多带来路径爆炸问题[48,49]。基于摘要的方法的本质是对程序中的代码块或数据结构的一种抽象，是在符号执行的过程中能用该抽象代替实际代码或数据的执行，这样能避免在抽象的代码块中作路径扩展，从而在一定程度上缓解路径爆炸问题[40-52]。

1. **目标导向的方法**

目标导向的方法通过设定目标程序实体(通常是程序中的危险语句或关键语句)，任何与到达该目标程序实体无关的路径都被排除。Ma等人引入两种符号执行算法[53]，分别是最短路轻符号执行和反向调用链符号执行,使得符号执行能够运行到指定语句。Yang等人提出了一种定向増量符号执行算法[54]，算法的场景是存在两个新旧不同版本的程序，而旧程序已经经过充分地测试，现在要使用符号执行对新版本的程序进行符号执行分析。该方法使用静态分析辅助符号执行，在此种应用场景下倾向选择能够到达的新程序的特定代码段的路径，能够有效缓解路径爆炸问题。

1. **程序中特定语法结构的处理方法**

此类方法依据程序中的特定的结构来缓解符号执行的路径爆炸问题。Godefroid等人根据特定程序的输入语法规则来构造测试用例，比如对于音视频解码器程序来说，其输入都有一定的格式，这样在使用符号执行构造测试用例时可利用输入的格式信息，这样能够大大减少约束求解器的解空间。Saxena等人提出的一种对循环的静态处理方法[55]，解决循环中丢失的变量和输入的控制流依赖问题，能有比一般的符号执行找到更多的漏洞的机会。传统的符号执行的搜索策略是无方向的穷举过程，而该方法可以通过生成关于循环无关变量的约束来快速找到漏洞，而不必穷举循环中的毎一条路径，所以在一定程度上能够缓解符号执行的路径爆炸问题。

**2) 系统调用和库函数调用**

当程序中存在系统调用时，进程从用户态进入运行态，当使用符号执行分析程序时一般的做法是将被检测进程嵌入到符号执行引擎进程的进程空间里去，而一般运行在用户态的符号执行引擎进程是无法追踪到内核态的；或者对于源码级的符号执行工具来说，库函数一般都是以二进制代码提供给程序的，当程序调用库函数时，因为源码符号执行需要获得源代码，此时符号执行便无法分析库函数中的代码。这样就会丢失内核代码和库函数代码中的符号约束，从而使符号执行进入不一致状态。

一种处理系统级调用的方法是为这些外部代码创建摘要，这些摘要用于描述不管程序以何种具体值调用外部代码所有可能的情况[56]。这种方法需要为每一个库函数和系统调用手动创建摘要，而且外部代码状况有可能无法穷举，因此该方法并不能很好的解决系统调用和库函数调用问题。

* 1. 研究背景与意义

在课题组前期的迭代蜕变测试研究工作中，发现原始测试用例的生成直接影响蜕变测试的故障检测能力。针对同一条蜕变关系，选用不同的原始测试用例进行蜕变测试，结果可能相差很大。本文从待测程序的路径覆盖角度展开研究，旨在生成高覆盖率的原始测试用例，进而提高蜕变测试的故障检测能力。

本文的研究工作得到国家自然科学基金（面上项目）“面向SOA 软件的蜕变测试技术研究”（项目编号：61370061，2014.01.01~2017.12.31）的资助。

* 1. 研究内容与成果

本文旨在从生成高效的原始测试用例的角度，对蜕变测试性能进行优化。结合符号执行和约束求解技术，提出一种原始测试用例自动生成方法。该方法能够根据待测程序的路径约束条件，生成满足相应路径约束的原始测试用例，进一步提高了蜕变测试的自动化程度。本文的主要贡献如下：

1. **提出了一种基于符号执行的原始测试用例生成技术**：从程序路径覆盖角度出发，提出一种高效的原始测试用例生成方法,提高了蜕变测试的故障检测能力。
2. **提出一种基于路径距离的原始测试用例优先级排序方法：**依据测试用例覆盖的程序路径距离，对原始测试用例进行优先级排序。在测试资源有限的情况下，该方法通过调整测试用例的执行次序进一步提高蜕变测试的故障检测能力。
3. **设计与实现了基于符号执行的原始测试用例生成技术的支持工具：**本文在蜕变测试工具MT4WS的原始测试用例生成模块中添加基于符号执行的生成方法，依据Web服务的源程序自动生成高覆盖率的原始测试用例，进一步提高了蜕变测试的自动化程度。
4. **采用实例程序评估了所提原始测试用例生成技术可行性与原始测试用例优先级排序方法的有效性：**采用实例研究的方式，验证了基于符号执行的原始测试用例生成技术可行性与有效性；评估了对基于路径距离的原始测试用例优先级排序方法的有效性。实验结果表明所提方法能有效的提高蜕变测试的故障检测能力。
   1. 论文组织结构

本文的组织结构如下：

1. 引言；
2. 背景介绍，包括本文中涉及的相关技术和工具，国内外研究现状，研究内容及成果和研究意义；
3. 深入讨论基于符号执行的原始测试例生成方法，包括对技术的原理描述，相关的定义和概念；并根据路径距离提出了一种原始测试用例优先级排序方法；
4. 讨论基于符号执行的原始测试用例生成技术的工具设计与实现，包括工具的需求分析，系统结构设计以及系统演示；
5. 对基于符号执行的原始测试用例生成技术的有效性进行实例研究，并实验验证了基于路径距离的原始测试用例优先级排序方法的有效性。包括实验问题描述、实验对象介绍、实验步骤、度量指标以及实验结果分析；
6. 结论。
7. 基于符号执行的原始测试用例生成技术

本章介绍基于符号执行的原始测试用例生成技术的原理以及实现过程;阐述基于路径距离的原始测试用例优先级排序方法，并给出算法描述，最后给出实例演示方法过程。

* 1. 基本思想

蜕变测试通过验证一组测试用例输出的结果是否满足蜕变关系来检测程序是否潜藏故障。对于同一条蜕变关系，不同测试用例的测试结果可能不同。在一组测试用例中，衍生测试用例的生成依赖于原始测试用例和蜕变关系，因此生成高效的原始测试用例有助于提高蜕变测试的故障检错能力。

程序的故障包含各种不同的特征，例如故障的位置、故障的类型等。若程序中潜藏着某个故障，选用的测试用例必须执行到存在错误的语句才有可能检测到这个故障。例如的程序。程序中的方法实现的是统计数组中值为100的个数。错误的语句在程序的第6行，因此，检测到这个程序的必要条件就是设计的测试用例能够执行到这行代码。

对于，若采用随机值法生成测试用例，即使生成的测试用例集很大，也很难覆盖到错误语句。在本文的研究方法中利用程序的路径约束条件信息指导测试用例的生成。将模型检测和符号执行相结合，获得程序所有执行路径的约束条件，通过约束求解器生成满足这些约束条件的测试用例集并以此作为蜕变测试中的原始测试用例。

|  |  |  |
| --- | --- | --- |
| 1 |  | int foo**(**int**[]** array**){** |
| 2 |  | int count**=**0**;** |
| 3 |  | int target**=**100**;** |
| 4 |  | **for(**int i**=**0**;**i**<**array**.**length**;**i**++){** |
| 5 |  | **if(**array**[**i**]** **==** target**){** |
| 6 |  | count**--;**//错误语句 |
| 7 |  | **}** |
| 8 |  | **}** |
| 9 |  | **return** count**;** |
| 10 |  | **}** |

图 3-1 示例程序

* 1. 相关定义及概念

下面定义了本文中用到的基本概念。

**定义1（路径集）** 对于给定的测试用例集*TC*，*TC*中所有测试用例在待测程序P中执行的路径集合称为*TC*对于*P*的路径集，用*path(TC, P)*表示。*path*(*TCi*, *P*)表示测试用例*TCi*在*P*中的执行路径（*TCi*为*TC*中的测试用例）, {*path*(*TC, P*)}表示路径集覆盖语句的集合。

**定义2（路径长度）** 对于路径*path*(*TCi, P*)，{*path*(*TCi, P*)}所覆盖的程序语句数称为路径集的路径集长度，用*s*(*TCi, P*)表示。计算公式如下：

*s*(*TCi, P*) = |{*path*(*TCi, P*)}|

**定义3（路径距离）** 对于给定的测试用例*tc*和测试用例集*TC*，所有存在于{*path*(*tc,P*)}并且不存在于{*path*(*TC,P*)}的程序语句数称为路径*path*(*tc,P*)相对于路径集*path*(*TC,P*)的路径距离，用*diff*(*path*(*tc,P*),*path*(*TC,P*))表示。计算公式如下:

*diff*(*path*(*tc,P*), *path*(*TC,P*)) = |{(*path*(*tc,P*)}-{*path*(*TC,P*)}|

**定义4（蜕变域）**[11]假设为程序的输入域,  ，在上可以对进行蜕变测试，则称为程序的蜕变域，记作，称为的非蜕变域，记作。

* 1. 原始测试用例生成

本节介绍基于符号执行的原始测试用例生成方法。描述了原始测试用例生成的原理。



图 3-2 原始测试用例生成原理

1. **提取每条程序路径的约束条件表达式**

为了获得程序的路径约束条件表达式，需要对程序进行符号执行，在本文中使用符号执行工具SPF分析Java的字节码文件并进行符号执行。另外，需要JPF构建程序运行时的状态空间，以便在符号执行时能探索程序所有的可行路径并收集相应的路径约束条件。

获取程序路径约束表达式的具体流程如下：

1. **配置符号执行信息**：为了进行符号执行，需要指定哪些参数是符号值，哪些参数是具体值。为了获取路径的约束表达式，需要配置监听器对虚拟机运行情况进行监听。
2. **创建符号变量**：根据配置信息，创建符号变量替换真实值在程序中运行。
3. **构建符号执行树**：使用符号变量执行程序，遍历系统可执行的路径，并记录每一条路径的约束条件，由此生成代表程序执行路径的符号执行树。
4. **获取路径约束条件表达式**：遍历符号执行树获取每一条路径的约束条件表达式。
5. **根据约束条件表达式生成原始测试用例**

提取出每条测试路径的约束条件表达式之后，需要生成满足这些约束条件的测试用例，在传统的方法中，约束条件表达式大部分是需要人工分析，这样不仅繁琐耗时，并且在项目相当大的时候，有可能造成条件的遗漏，可靠性不能保障；除人工分析外，还有一种方法是随机生成大量的测试用例，从中选择满足约束条件的测试用例。由于随机生成的数据往往具有不确定性，对于路径约束条件比较难以满足的路径，很难随机生成相应的测试用例。并且从随机生成的测试用例中选取满足约束条件的测试用例本身的成本也很高。

本文采用约束求解工具choco对生成的约束条件表达式进行求解，生成执行相应路径的测试用例。具体的实现包括问题模型创建和约束求解两部分。

1. **问题模型创建**：choco提供创建问题模型的API，创建一个Model对象对问题进行建模。读取约束条件表达式，根据表达式中变量的类型创建相应的变量并添加到Model中并根据约束条件添加变量与变量之间的约束关系。
2. **约束求解**：choco提供了一个针对问题模型进行求解的Solver对象，根据Model对象中的变量类型以及变量与变量之间约束关系，可以对问题模型进行求解以获得满足约束条件的数据。

同样的步骤处理每条测试路径生成的约束条件表达式,可以求解出满足对应程序路径提取出的约束条件表达式的可行解,即生成满足高覆盖率的测试用例集。需要注意的是这些测试用例并不能直接作为蜕变测试的原始测试用例。根据定义4，不同的蜕变关系有不同的蜕变域，因此用上述生成的测试用例集进行蜕变测试的时候需要先判断测试用例是否满足蜕变关系的蜕变域。

* 1. 原始测试用例优先级排序

在测试资源有限的情况下，需要对测试用例进行精简，用尽量少的测试用例检测出尽可能多的故障。针对于同一个待测程序，不同的测试用例的故障检测能力可能差别很大，为了能在精简测试用例的情况下，检测到尽可能多的故障，有必要将不同的测试用例根据故障检测能力进行排序，优先执行故障检测能力高的测试用例。

根据3.3节的方法得到的原始测试用例集和原始测试用例相对应的路径执行轨迹，提出一种原始测试用例优先级排序方法，是基于路径距离的原始测试用例优先级排序的算法描述。

|  |  |
| --- | --- |
| **Algorithm: Sorting method for source test cases based on**  **paths information** | |
| **Input**：unordered source test case suite **TC** and  corresponding paths **Paths** | |
| **Output**：source test case sorted by priority **PrioritizedTC** | |
| 1 | **Initialize** PrioritizedTC = PrioritizedPaths = ∅ |
| 2 | **while** TC ≠ ∅ |
| 3 | **Initialize** max = 0 index = 0 |
| 4 | **for**  i ← 1 to Paths.size( ) **do** |
| 5 | **if** diff(PrioritizedPaths，Paths[i]) >max |
| 6 | max = diff(PrioritizedPaths，Paths[i]) |
| 7 | index = i |
| 8 | **end if** |
| 9 | **end for** |
| 10 | add Paths[index] to PrioritizedPaths |
| 11 | remove Paths[index] from Paths |
| 12 | add TC[index] to PrioritizedTC |
| 13 | remove TC[index] from TC |
| 14 | **end while** |
| 15 | **return** PrioritizedTC |

图 3-3 基于路径距离的原始测试用例优先级排序算法

基于路径距离原始测试用例优先级排序方法的具体过程如下：

1. 依据待测程序的源文件，结合模型检测和符号执行，获取程序的路径约束条件和路径执行轨迹。
2. 根据路径约束条件，通过约束求解获得满足路径约束的测试用例集。
3. 将测试用例依据路径长度进行排序，选择路径长度最大的测试用例，放入已排序的测试用例集中。
4. 从未排序的测试用例集中选出一个与已排序的测试用例集路径距离最大的测试用例并添加到已排序的测试用例集中。
5. 循环执行步骤4)直到未排序的测试用例集为空，得到按优先级排序的测试用例集。
   1. 方法示例

本节用一个简单的程序来阐述基于符号执行方法生成原始测试用例的过程。是一个简单的多分支程序，程序中有两个输入变量x和y。根据不同的输入值，程序会执行不同的路径。

|  |  |  |
| --- | --- | --- |
| 1 |  | public class MyClass**{** |
| 2 |  | public int myMethod**(**int x**,** int y**){** |
| 3 |  | int z **=** x **+** y**;** |
| 4 |  | **if** **(**z **>** 0**)** **{** |
| 5 |  | **if(**y**>**0**){** |
| 6 |  | z **=** 1**;** |
| 7 |  | **}** **else{** |
| 8 |  | z **=** **-**1**;** |
| 9 |  | **}** |
| 10 |  | **}** **else** **{** |
| 11 |  | **if(**x**>**0**){** |
| 12 |  | z **=** z **-** x**;** |
| 13 |  | **}** **else{** |
| 14 |  | z **=** z **+** x**;** |
| 15 |  | **}** |
| 16 |  | **}** |
| 17 |  | z **=** 2 **\*** z**;** |
| 18 |  | **return** z**;** |
| 19 |  | **}** |
| 20 |  | **}** |

图 3-4 多分支示例程序

上述多分支示例程序的程序流程图如所示，从程序流程图中可以分析出程序中存在四条可能执行的路径。



图 3-5 多分支示例程序流程图

下面具体阐述根据程序路径约束信息生成原始测试用例的过程：

1. **程序符号执行**：将程序的输入值用符号值替换，用符号x\_1\_SYMINT替换输入变量x, y\_2\_SYMINT替换输入变量y，在特定的虚拟机环境下执行程序，在系统状态空间模型上遍历可执行路径并获取路径约束条件和路径执行轨迹，获取的路径信息如表3-1所示：
2. **路径约束条件求解**：根据上一步得到的路径约束条件，进行约束求解，可以获得满足执行对应路径的测试用例。生成的测试用例如表3-2所示。
3. **原始测试用例排序**：针对生成的原始测试用例，用3.4节中的优先级排序方法进行排序，排序结果如表3-3所示。表中优先级表示原始测试用例的执行的顺序。其中优先级1表示优先级最高。

通过上述三个步骤，获得了高覆盖率的测试用例，并以此作为蜕变测试中的原始测试用例，能够有效的提高蜕变测试的故障检测能力。

表 3-1 多分支示例程序路径信息

|  |  |
| --- | --- |
| **序号** | **路径信息** |
| 1 | <Constraint>  y\_2\_SYMINT> CONST\_0 && (y\_2\_SYMINT+ x\_1\_SYMINT) > CONST\_0  </Constraint>  <Path>  MyClass\_3 -> MyClass\_4 -> MyClass\_5 -> MyClass\_6-> MyClass\_17-> MyClass\_18  </Path> |
| 2 | <Constraint>  y\_2\_SYMINT <= CONST\_0 && (y\_2\_SYMINT + x\_1\_SYMINT) > CONST\_0  </Constraint>  <Path>  MyClass\_3 -> MyClass\_4 -> MyClass\_5 -> MyClass\_7 -> MyClass\_8 -> MyClass\_17 -> MyClass\_18  </Path> |
| 3 | <Constraint>  x\_1\_SYMINT > CONST\_0 && (y\_2\_SYMINT + x\_1\_SYMINT) <= CONST\_0  </Constraint>  <Path>  MyClass\_3 -> MyClass\_4 -> MyClass\_10 -> MyClass\_11 -> MyClass\_12 -> MyClass\_17 -> MyClass\_18  </Path> |
| 4 | <Constraint>  x\_1\_SYMINT<=CONST\_0 && (y\_2\_SYMINT + x\_1\_SYMINT) <= CONST\_0  </Constraint>  <Path>  MyClass\_3 -> MyClass\_4 -> MyClass\_10 -> MyClass\_11 -> MyClass\_13 -> MyClass\_14 -> MyClass\_17 -> MyClass\_18  </Path> |

注：<Constraint>标签内表示路径的约束条件表达式，其中CONST\_0表示常量0，<Path>标签表示路径执行轨迹，用该路径执行的语句行号表示。

表 3-2 测试用例生成结果

|  |  |
| --- | --- |
| **序号** | **测试用例** |
| 1 | x=0,y=1 |
| 2 | x=2,y=-1 |
| 3 | x=1,y=-1 |
| 4 | x=-1,y=-1 |

表 3-3 测试用例优先级排序结果

|  |  |  |
| --- | --- | --- |
| **序号** | **测试用例** | **优先级** |
| 1 | x=0,y=1 | 3 |
| 2 | x=2,y=-1 | 2 |
| 3 | x=1,y=-1 | 4 |
| 4 | x=-1,y=-1 | 1 |

1. 基于符号执行的原始测试用例生成工具的设计与实现

本章针对基于符号执行的原始测试用例生成技术工具的设计分析与实现展开讨论，包括工具的需求分析和系统结构设计,最后用一个实例演示工具的使用。该工具目前已经集成到课题组已有的蜕变测试工具MT4WS中。

* 1. 需求分析

MT4WS目前支持的原始测试用例生成方法包括随机值法、文件导入法和手工输入法。针对基于符号执行的原始测试用例生成方法的功能需求为：系统需要根据Web服务的源程序，生成高覆盖率的原始测试用例集并对该Web服务进行蜕变测试。

描述了MT4WS系统的用例图，其中详细的描述了原始测试用例生成模块。



图 4-1 MT4WS中原始测试用例生成模块用例图

下面将对进行具体的描述：

1. **选择待测Web服务**

测试人员输入Web服务的WSDL URI，系统将解析该WSDL。WSDL URI可以是URL或者是本地文件路径。

1. **选择待测的Operation**

一个Web服务可能包含有多个功能，用户需要选择对哪一个功能进行蜕变测试，待选的功能Operation由系统自动解析WSDL获得。

1. **管理蜕变关系集**

蜕变关系集的管理包括生成蜕变关系、删除蜕变关系和蜕变关系的检查。其中蜕变关系的生成方式又包括手工输入蜕变关系、文件导入蜕变关系和基于数据变异获取蜕变关系3种方法。由于篇幅原因，在中没有体现。

1. **选择原始测试用例生成方式**

测试人员需要选择某种方法生成原始测试用例，目前系统支持支持随机生成原始测试用例、手工输入原始测试用例、文件导入原始测试用例，另外，本文中提出的基于符号执行的原始测试用例生成技术也作为一个功能模块集成到MT4WS中。

1. **测试配置信息管理**

测试人员可以选择查看或者修改测试配置信息，如选择执行测试的测试用例集、补充测试用例等。

1. **执行测试**

用户通过MT4WS客户端进行测试。

1. **查看测试报告**

用户完成测试之后，可以在客户端界面上查看测试报告，另外系统支持下载完整的测试报告。

通过中划线的部分可以得到系统的明确需求：系统需要添加基于符号执行的原始测试用例生成方式。进一步对需求进行细分得到以下4个功能需求：

1. **选择配置文件**

测试人员选择配置文件的路径，配置文件中包含了Web服务源文件所在位置，符号执行和约束求解的配置信息，如程序中方法的哪些变量为符号值，哪些变量为具体值。

1. **创建程序状态空间模型**

为了能够遍历程序中所有可执行的路径，需要对程序进行状态空间模型构建，以便程序在运行时能在状态空间内进行状态匹配，回溯等。

1. **生成路径约束条件**

在源程序上进行符号执行，用符号值代替真实值运行程序，在执行每条程序路径时候收集路径的约束条件，最后生成约束条件表达式。

1. **路径约束求解**

根据路径约束条件表达式，可以用约束求解器获得满足执行对应路径的测试用例，对程序所有的路径约束条件表达式进行求解可以获得满足高覆盖率的测试用例。

* 1. 系统设计

根据4.1节中的需求分析，本节将详细介绍基于符号执行的原始测试用例生成工具的系统结构设计。

为MT4WS的系统框架图，本文实现的原始测试用例生成方法的功能将集成于该工具中。基于符号执行的原始测试用例生成器位于系统的原始测试用例生成器中。该系统首先通过WSDL解析器解析待测Web服务的WSDL文档得到待测服务所提供的功能，以及对应的输入与输出参数的描述；蜕变关系定义器生成蜕变测试中需要的蜕变关系，目前系统实现了三种蜕变关系生成方法；测试用例生成器有四种定义方法，其中自动生成器根据程序的源文件解析程序所有路径的约束条件表达式，通过约束求解获得原始测试用例。衍生测试用例是通过蜕变关系和原始测试用例生成的,生成的测试用例将存入数据库中；执行器通过调用服务实现，进行蜕变测试过程，该过程需要分别执行原始测试用例和衍生测试用例得到Web服务输出，同时执行器执行之前可以调用配置器进行配置操作；评估器根据原始测试用例和衍生测试用例的执行结果是否满足相应蜕变关系判定测试是否通过，评估结束后会生成测试日志文档。



图 4-2 MT4WS系统框架图

为基于符号执行的原始测试用例生成器的系统框架。

1. **测试路径生成器**

测试路径生成器根据程序源文件解析程序中可执行的路径，进行符号执行，获取路径约束条件表达式。

1. 状态空间模型创建

* 程序状态生成：根据程序源文件，在JPF提供的特殊虚拟机上运行程序，通过JVM内置的程序状态生成器生成程序当前运行的状态，程序的状态包括每个线程的信息、类的静态变量、对象的动态变量。
* 程序状态管理：在JPF特殊的虚拟机上可以对程序状态进行管理，这样可以保证状态搜索的顺利进行。在虚拟机中提供了三个方法：前进、回溯和状态恢复。前进用于生成下一个程序状态，报告生成的状态是否有后续，如果有，则保存在回溯栈中，以便恢复；回溯用于从回溯栈中恢复到上一个程序状态；状态恢复用于恢复到任意的程序状态。
* 系统状态模型：系统状态模型类似于一个由节点和有向边组成的图，图中节点表示程序的状态，有向边表示程序状态之间的转换，这样程序就用一种抽象的形式来表示。在系统状态模型中，可以用不同的搜索算法遍历模型中的状态执行不同的程序路径。

1. 路径信息获取

在虚拟机上执行字节码文件的时候，可以用符号值代替真实值去运行程序，在程序状态中变量的值也用符号表达式表示。通过遍历系统状态模型中的路径，获取程序中可执行路径的信息，路径信息包括路径执行轨迹和路径约束条件。

1. **测试数据生成器**

测试数据生成器负责提取每条测试路径的约束条件表达式，利用约束求解获取满足条件的可行解，即针对每条测试路径的测试用例。

1. 问题模型创建

问题模型是对需要求解的约束条件表达式的一个抽象,问题模型创建包括约束条件提取和创建变量两个模块。针对每一条测试路径，从测试路径信息中提取出所有的约束条件，根据表达式中变量的类型创建变量实例，在问题模型中添加路径约束条件表达式中变量与变量之间的约束。

1. 数据生成

根据路径约束表达式创建了问题模型之后，可以对问题进行求解，求得的可行解即为满足相应约束条件表达式的数据。针对每条测试路径生成约束表达式的可行解，这些可行解数据组成了测试用例集。



图 4-3 测试用例自动生成器结构图

* 1. 工具详细设计

工具实现的关键问题是设计合理的数据存储结构，用于描述路径和路径约束表达式之间的关系，同时要满足可扩展性。图4-4描述了路径信息的存储结构。



图 4-4 路径信息存储结构

1. Comparator类

该类为一个枚举类，包含了六种变量与变量之间常见的约束关系：“= =”、“!=”、“<”、“>”、“<=”、“>=”。

1. Constraint类

该类用于描述一个约束关系，其中成员变量left和right用于表示一个变量或者表达式，它们之间的关系用Comparator类型的com来描述。Constraint是一种递归的结构，包含了一个本类类型的成员变量and用于表示约束表达式末尾的约束条件。

1. PathCondition类

该类用于描述路径约束条件，包含一个Constraint类型的成员header表示路径的约束条件表达式，count则表示路径约束的个数。该类的solve( )方法根据本类实例对象中header约束条件表达式求解满足约束的可行解。

1. Path类

该类用于描述程序执行的一条路径，其中成员变量pc表示该路径的路径约束条件，成员变量path表示路径的执行轨迹，length代表路径覆盖的语句数。

* 1. 系统演示

本节用航空行李托运计费服务作为实例，演示MT4WS的功能，其中原始测试用例的生成采用基于符号执行的方法。

1. 解析WSDL

为系统的欢迎界面，在界面中输入待测Web服务的WSDL URI，点击Start按钮，系统会对WSDL文档进行解析。

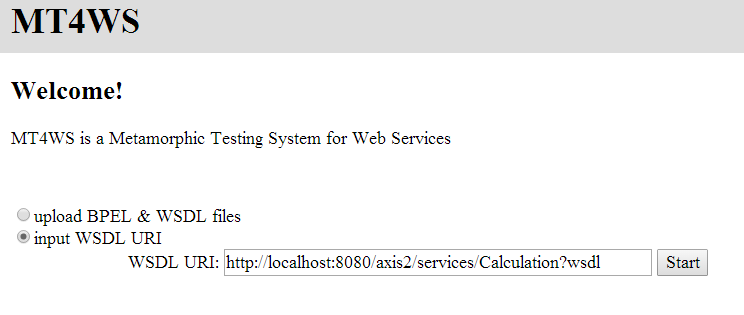


图 4-5 MT4WS欢迎界面

1. 选择待测接口

是系统Operation选择界面。Operation List是根据WSDL文档解析出的Web服务包含的所有的功能接口。在本次演示中选择feeCalculation接口进行测试，点击Next进入蜕变关系定义界面。

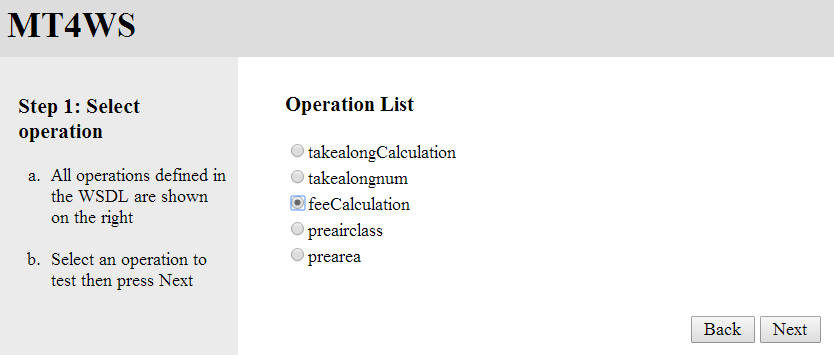


图 4-6 Operation选择界面

1. 蜕变关系定义

是系统的蜕变关系定义界面。界面中三个单选框代表了三种蜕变关系的定义方法，定义完一条蜕变关系后，点击Add to MR set将定义的蜕变关系添加到蜕变关系集中。点击Next进入下一步。

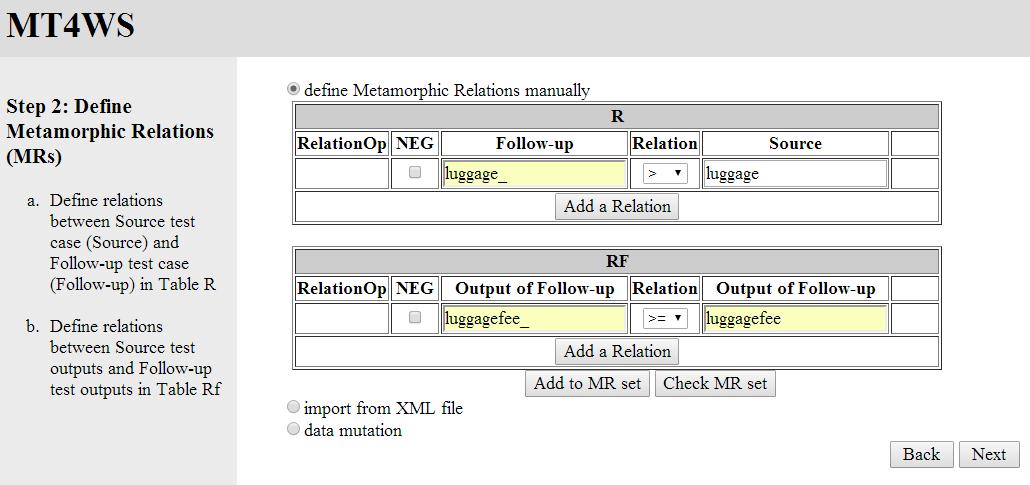


图 4-7 蜕变关系定义界面

1. 原始测试用例生成

为系统的原始测试用例生成界面，系统目前支持四种原始测试用例生成方式，分别用四个点选框表示。选择Generate from source file，采用基于符号执行的方法生成原始测试用例。为了更好集成JPF和SPF，系统采用JPF的配置文件格式。点击Next系统根据配置文件信息生成原始测试用例，并添加到数据库中。

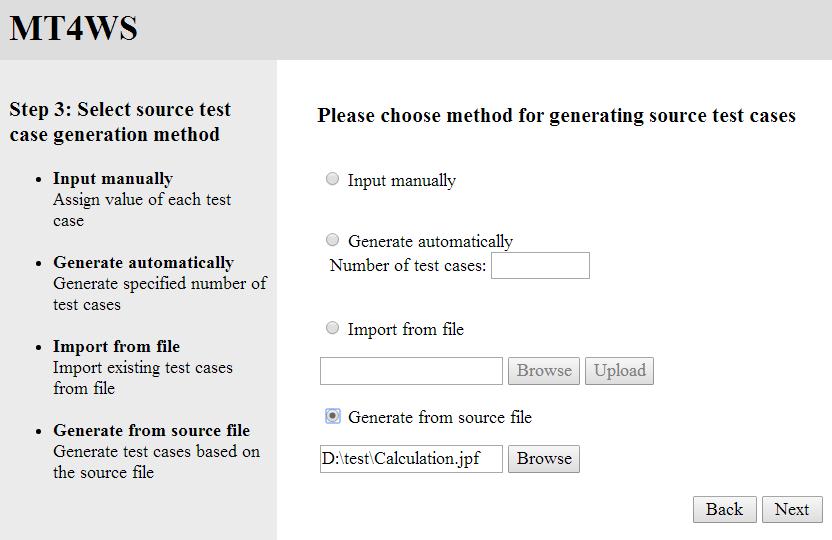


图 4-8 原始测试用例生成界面

1. 测试配置

为测试配置界面，界面中显示当前测试Web服务的名称、WSDL URI以及待测接口名称。另外，系统还支持蜕变关系管理和测试用例配置。

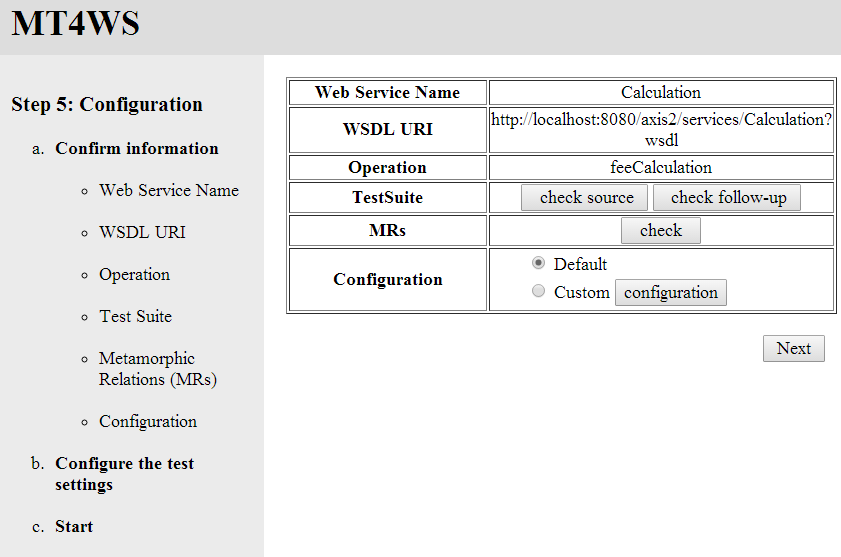


图 4-9 测试配置界面

显示了蜕变关系集的管理界面，用户可以编辑或删除蜕变关系。

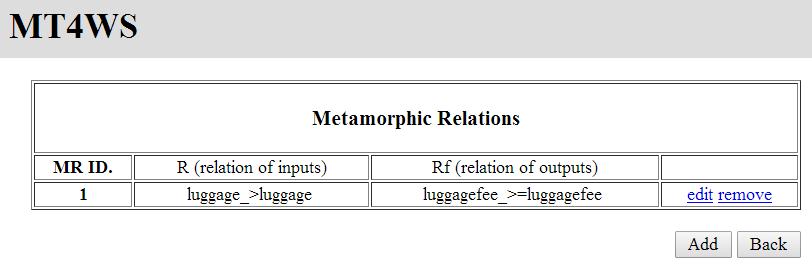


图 4-10 蜕变关系集管理界面

显示了测试用例的配置界面，用户可以选择部分的测试用例进行蜕变测试，另外，系统还支持两种方式补充测试用例，分别是随机生成和根据蜕变关系迭代生成。

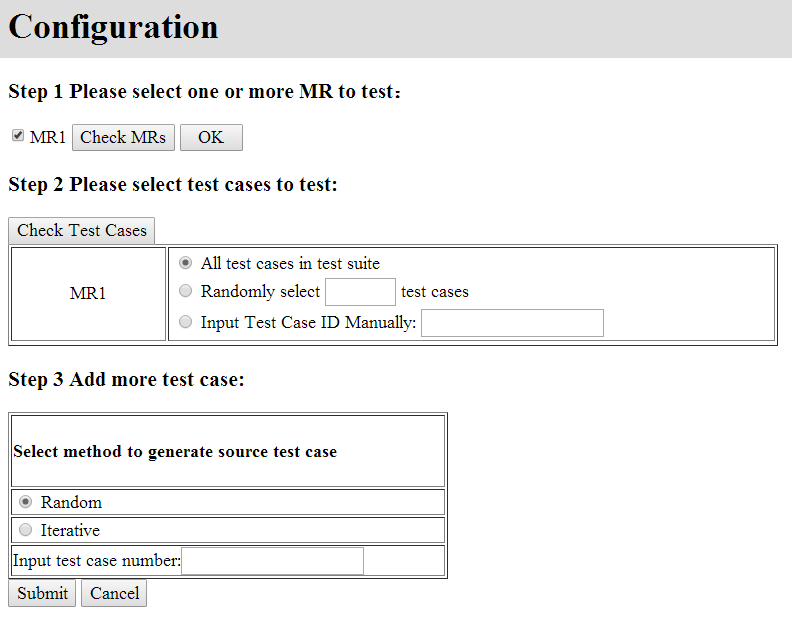


图 4-11 测试用例配置界面

为系统的执行测试界面，该界面显示测试正在进行，测试完成后自动跳转到测试报告界面。

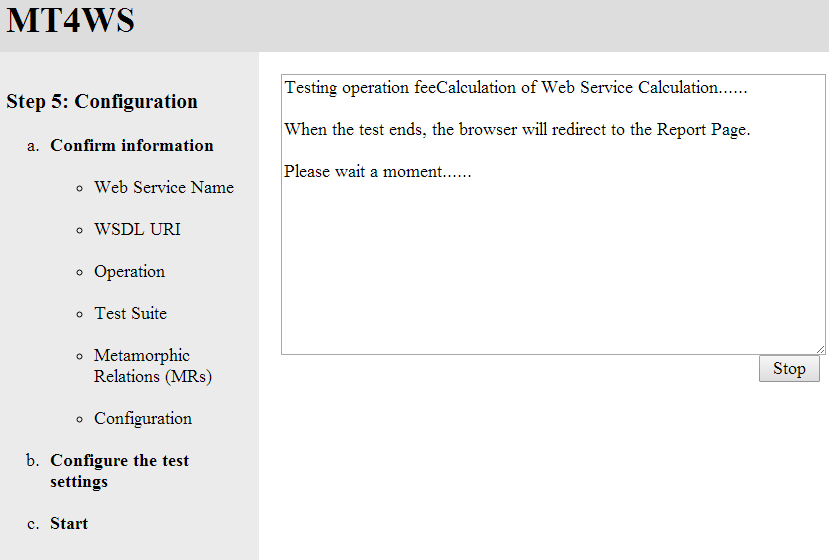


图 4-12 测试执行界面

1. 测试报告

为系统测试报告界面，界面中显示了待测Web服务的基本信息、接口信息以及测试结果。

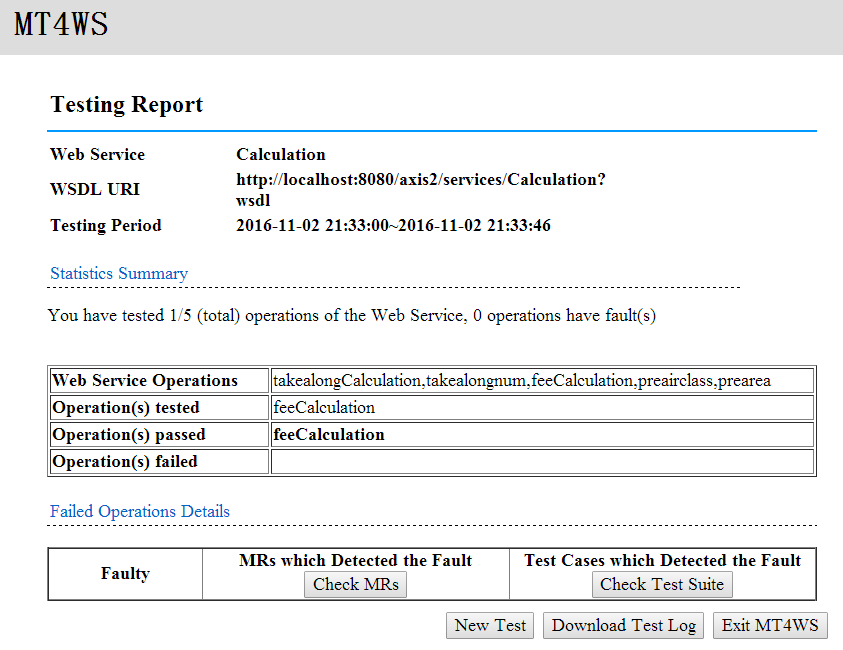


图 4-13 测试报告界面

* 1. 本章小结

本章详细讨论了基于符号执行的原始测试用例生成技术工具的设计与实现，并集成到课题组已有的蜕变测试工具MT4WS中。工具可以根据Web服务源程序生成高覆盖率的原始测试用例，进一步提高了蜕变测试的自动化程度。

1. 实验评估

本章选择三个java程序进行实例研究: （1）航空行李托运计费程序；（2）中国联通计费程序；（3）停车计费程序。验证本文提出方法的可行性和有效性。三个实例程序均根据真实的规格说明，由课题组同学开发。

* 1. 研究问题

本章的实例研究围绕三个问题展开讨论。

1. **验证基于符号执行的原始测试用例生成技术的可行性与适用性。**
2. **验证基于符号执行的原始测试用例生成技术的有效性。**实验中将针对相同的蜕变关系，分别用上述方法和随机值法生成原始测试用例进行蜕变测试，比较两种原始测试用例生成方法的故障检测能力。
3. **验证基于路径距离的原始测试用例优先级排序方法的有效性。**实验中分别从按优先级排序的原始测试用例集和未排序的原始测试用例集分别选取相同比例的原始测试用例，在相同蜕变关系集的条件下，比较两组选取的原始测试用例集进行蜕变测试的故障检测能力。选取比例设定为10%，20%，30%，40%，50%，60%，70%，80%，90%。
   1. 实验步骤

针对上面的三个研究问题，设计两组对比实验分别进行研究。

为了验证基于符号执行的原始测试用例生成技术的可行性和有效性，设计一组对比实验。在获取相同的蜕变关系，生成相同变异体的条件下，比较上述方法和随机值法生成原始测试用例的故障检测能力。具体的实验步骤如下：

1. **获取蜕变关系**：3个实例均采用了基于数据变异的蜕变关系获取技术[24],并结合程序的属性设计蜕变关系。
2. **生成变异体：**实验中采用变异分析工具MuJava为待测程序生成变异体，其中包含所有典型的变异算子，并已去除等价变异体。
3. **生成原始测试用例：**实验中测试用例的生成是对比实验的关键问题，采用了两种不同的测试用例生成策略：(1)基于符号执行的原始测试用例生成方法——实验中根据程序的源文件，解析程序的所有路径，根据路径约束条件，生成满足路径约束条件的原始测试用例。(2)随机值法——根据程序的输入规格随机生成测试用例，实验中设定TC=n、TC=2n、TC=5n、TC=10n四组原始测试用例集(其中n代表方法(1)生成的原始测试用例数量)。
4. **判断原始测试用例是否满足蜕变域**：针对步骤3)生成原始测试用例集，分别判断是否满足步骤1)获取的蜕变关系集，如果满足，根据蜕变关系为原始测试用例生成相应的衍生测试用例，并添加到待测测试用例集中。
5. **执行测试：**将原始测试用例集和对应的衍生测试用例集分别在步骤2)生成的非等价变异体上执行，比较原始测试用例的输出结果和衍生测试用例的输出结果是否满足蜕变关系，得到测试结果。
6. **结果分析**：实验结果采用变异分析技术进行分析，度量指标选用变异得分(Mutation Score)。变异得分指的是用测试用例集TC能够杀死程序P的变异体数量占生成非等价变异的数量的百分比(测试用例集能够检测到变异体的故障称为变异体被杀死)。计算公式如下:



其中表示被杀死的变异数量，Nm表示总的变异体数量，表示等价变异体数量。变异得分能有效的反应测试用例集的故障检测能力。

为了验证基于路径距离的原始测试用例优先级排序方法的有效性，同样设计一组对比实验。实验步骤与第一组对比实验除步骤3)之外都相同，在本实验中，需要先设定测试用例选取比例，分别为10%，20%，30%，40%，50%，60%，70%，80%，90%。采用基于符号执行的方法生成原始测试用例集并根据路径距离进行排序，从按优先级排序的原始测试用例集和未排序的原始测试用例集中分别选取相同比例的原始测试用例进行蜕变测试。

* 1. 实验对象

1. **航空行李托运计费程序**

航空行李托运计费程序根据乘客座舱等级、航班信息、是否为学生、行李重量以及机票价格计算行李的托运费，托运费用的计算参照中国国际航空公司行李托运计费标准，程序采用Java实现。

1. **输入规格说明**

描述了航空行李托运计费程序的输入规格，其中airClass值0、1、2、3分别代表乘客的座舱等级为头等舱、公务舱、经济舱和婴儿票。area值0、1分别代表国内和国际航班。isStudent表示是否为学生票，luggage表示用户所携带的行李总重量，economicfee表示用户所购买的飞机票价。

表 5-1 航空行李托运计费程序输入参数描述

|  |  |  |
| --- | --- | --- |
| 输入参数 | 参数类型 | 限制 |
| airClass | int | 仅为数字0-3 |
| area | int | 仅为数字0-1 |
| isStudent | boolean |  |
| luggage | double |  |
| economicfee | double |  |

1. **输出规格**

航空行李托运计费程序的输出参数为*luggagefee*，具体计算公式如下：



在上式中，*luggagefee*是乘客应付的行李托运费用，如果乘客的行李在超限范围之内，*luggagefee＝0*；*benchmark*是行李超重的标准量，系统将根据乘客的座舱等级和身份确定具体的*benchmark*。具体如所示：

表 5-2 航空行李托运计费程序benchmark设置

|  |  |
| --- | --- |
| airClass | benchmark |
| 0 | 40 |
| 1 | 30 |
| 2 | 20 |
| 3 | 0 |

说明：在中还有一种特殊情况没有列出来，即当乘客为国际乘客且为学生时，benchmark＝30。我们假设所有学生均持经济舱机票。

1. **蜕变关系获取**

在本实例中蜕变关系的获取采用基于数据变异的蜕变关系获取技术[24]结合等价类划分生成36条蜕变关系，描述了生成的36条蜕变关系。

表 5-3 航空行李托运计费程序蜕变关系集

|  |  |  |
| --- | --- | --- |
| **蜕变关系**  **序号** | ***R*** | ***Rf*** |
| 1. airClass=0， isStudent=false，area=0~1，固定economicfee的值为1000 | | |
| MR1 | luggage\_=luggage+1, (40<=luggage<200) | luggagefee\_>luggagefee |
| MR2 | luggage\_=luggage-1, (41<=luggage<200) | luggagefee\_<luggagefee |
| MR3 | luggage\_=2\*luggage, (40<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR4 | luggage\_=0.5\*luggage, (80<=luggage<200) | luggagefee<=0.5\*luggagefee |
| 2. airClass=1， isStudent=false，area=0~1，固定economicfee的值为1000 | | |
| MR5 | luggage\_=luggage+1, (30<=luggage<200) | luggagefee\_>luggagefee |
| MR6 | luggage\_=luggage-1, (31<=luggage<200) | luggagefee\_<luggagefee |
| MR7 | luggage\_=2\*luggage, (30<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR8 | luggage\_=0.5\*luggage, (60<=luggage<200) | luggagefee<=0.5\*luggagefee |
| 3. airClass=2， isStudent=false，area=0~1，固定economicfee的值为1000 | | |
| MR9 | luggage\_=luggage+1, (20<=luggage<200) | luggagefee\_>luggagefee |
| MR10 | luggage\_=luggage-1, (21<=luggage<200) | luggagefee\_<luggagefee |
| MR11 | luggage\_=2\*luggage, (20<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR12 | luggage\_=0.5\*luggage, (40<=luggage<200) | luggagefee<=0.5\*luggagefee |
| 4. airClass=3， isStudent=false，area=0~1，固定economicfee的值为1000 | | |
| MR13 | luggage\_=luggage+1, (0<=luggage<200) | luggagefee\_>luggagefee |
| MR14 | luggage\_=luggage-1, (1<=luggage<200) | luggagefee\_<luggagefee |
| MR15 | luggage\_=2\*luggage, (0<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR16 | luggage\_=0.5\*luggage, (0<=luggage<200) | luggagefee<=0.5\*luggagefee |

表 5-3 航空行李托运计费程序蜕变关系集（续表）

|  |  |  |
| --- | --- | --- |
| **蜕变关系**  **序号** | ***R*** | ***Rf*** |
| 5. airClass=0， isStudent=true，area=0，固定economicfee的值为1000 | | |
| MR17 | luggage\_=luggage+1, (40<=luggage<200) | luggagefee\_>luggagefee |
| MR18 | luggage\_=luggage-1, (41<=luggage<200) | luggagefee\_<luggagefee |
| MR19 | luggage\_=2\*luggage, (40<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR20 | luggage\_=0.5\*luggage,(80<=luggage<200) | luggagefee<=0.5\*luggagefee |
| 6. airClass=1， isStudent=true，area=0，固定economicfee的值为1000 | | |
| MR21 | luggage\_=luggage+1, (30<=luggage<200) | luggagefee\_>luggagefee |
| MR22 | luggage\_=luggage-1, (31<=luggage<200) | luggagefee\_<luggagefee |
| MR23 | luggage\_=2\*luggage, (30<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR24 | luggage\_=0.5\*luggage,(60<=luggage<200) | luggagefee<=0.5\*luggagefee |
| 7. airClass=2， isStudent=true，area=0，固定economicfee的值为1000 | | |
| MR25 | luggage\_=luggage+1, (20<=luggage<200) | luggagefee\_>luggagefee |
| MR26 | luggage\_=luggage-1, (21<=luggage<200) | luggagefee\_<luggagefee |
| MR27 | luggage\_=2\*luggage, (20<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR28 | luggage\_=0.5\*luggage,(40<=luggage<200) | luggagefee<=0.5\*luggagefee |
| 8. airClass=3， isStudent=true，area=0，固定economicfee的值为1000 | | |
| MR29 | luggage\_=luggage+1, (0<=luggage<200) | luggagefee\_>luggagefee |
| MR30 | luggage\_=luggage-1, (1<=luggage<200) | luggagefee\_<luggagefee |
| MR31 | luggage\_=2\*luggage, (0<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR32 | luggage\_=0.5\*luggage, (0<=luggage<200) | luggagefee<=0.5\*luggagefee |

表 5-3 航空行李托运计费程序蜕变关系集（续表）

|  |  |  |
| --- | --- | --- |
| **蜕变关系**  **序号** | ***R*** | ***Rf*** |
| 9. airClass=2， isStudent=true，area=1，固定economicfee的值为1000 | | |
| MR33 | luggage\_=luggage+1, (30<=luggage<200) | luggagefee\_>luggagefee |
| MR34 | luggage\_=luggage-1, (31<=luggage<200) | luggagefee\_<luggagefee |
| MR35 | luggage\_=2\*luggage, (30<=luggage<100) | luggagefee\_>=2\*luggagefee |
| MR36 | luggage\_=0.5\*luggage,(60<=luggage<200) | luggagefee<=0.5\*luggagefee |

其中，“X”表示原始测试用例变量，“X\_”表示对应的衍生测试用例变量。

1. **变异体与测试用例生成**

应用MuJava工具对航空行李托运程序生成56个非等价变异体。

根据3.3节中的测试用例生成方法，为航空行李托运计费程序生成原始测试用例，共生成34个原始测试用例。应用随机值法生成原始测试用例集规模分别为TC=34，TC=68，TC=170，TC=340。

1. **中国联通计费程序**

中国联通计费程序是根据中国联通公司手机话费收费标准所实现的，程序可根据用户办理的套餐、套餐月费、通话时长、使用的流量以及视频电话时长计算用户当月应缴纳的话费。程序采用Java实现。

1. **输入规格描述**

表 5-4 中国联通计费程序输入规格说明

|  |  |  |
| --- | --- | --- |
| 输入参数 | 参数类型 | 限制 |
| planType | String | 只允许字符A,B |
| planFee | int |  |
| talkTime | int |  |
| flow | int |  |
| callViewTime | int |  |

说明：在中，planType代表的用户的套餐类型，套餐类型只能为A或者B；planFee代表套餐月费，根据选择的套餐类型分成不同的几种套餐月费；talkTime代表语音通话时间；flow代表当月使用的流量；callViewTime代表视频通话时间。

1. **输出规格描述**

联通计费程序的输出参数为bill，为double型，具体的计算公式为：



其中， *talkTimePer*是每分钟通话费用， *flowPer*是每分钟流量的费用，*talkTimeBench*和*flowBench*是套餐中赠送的通话时间和流量，*planFee*是用户每月所预定的基础套餐费，具体如所示：

表 5-5 中国联通计费程序计费标准说明

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| planType | planFee | talkTimeBench | flowBench | talkTimePer | flowPer |
| A | 46 | 50 | 150 | 0.25 | 0.3 |
| 96 | 96 | 240 | 0.15 | 0.3 |
| 286 | 286 | 900 | 0.15 | 0.3 |
| 886 | 3000 | 3000 | 0.15 | 0.3 |
| B | 46 | 120 | 40 | 0.25 | 0.3 |
| 96 | 450 | 80 | 0.15 | 0.3 |
| 126 | 680 | 100 | 0.15 | 0.3 |
| 186 | 1180 | 150 | 0.15 | 0.3 |

1. **蜕变关系描述**

在本实例中蜕变关系的获取采用基于数据变异的蜕变关系获取技术[24]结合等价类划分生成32条蜕变关系，根据程序的属性设计4条蜕变关系，共36条蜕变关系如所示。

1. **变异体与测试用例生成**

应用MuJava工具对中国联通话费计费服务进行变异体生成112个非等价变异体。

根据3.3节中的原始测试用例生成方法，为中国联通计费程序生成32个原始测试用例。应用随机值法生成原始测试用例集规模分别为TC=32，TC=64，TC=160，TC=320。

表 5-6 中国联通计费程序蜕变关系描述

|  |  |  |
| --- | --- | --- |
| **蜕变关系**  **序号** | ***R*** | ***Rf*** |
| 1.Type=A，planFee=46，50<=talkTime<=8000 & flow<=150 | | |
| MR1 | talkTime\_=talkTime+1,(50<=talkTime<=7999) | bill\_>bill |
| MR2 | talkTime\_=talkTime-1, (51<=talkTime<=8000) | bill\_<bill |
| MR3 | talkTime\_=2\*talkTime, (50<=talkTime<=4000) | bill\_>=bill |
| MR4 | talkTime\_=talkTime/2, (100<=talkTime<=8000) | bill\_<=bill |
| 2. Type=A，planFee=96，96<=talkTime<=8000 & flow<=240 | | |
| MR5 | talkTime\_=talkTime+1,(96<=talkTime<=7999) | bill\_>bill |
| MR6 | talkTime\_=talkTime-1, (97<=talkTime<=8000) | bill\_<bill |
| MR7 | talkTime\_=2\*talkTime, (96<=talkTime<=4000) | bill\_>=bill |
| MR8 | talkTime\_=talkTime/2, (96\*2<=talkTime<=8000) | bill\_<=bill |
| 3. Type=A，planFee=286，286<=talkTime<=8000 & flow<=900 | | |
| MR9 | talkTime\_=talkTime+1,(286<=talkTime<=7999) | bill\_>bill |
| MR10 | talkTime\_=talkTime-1, (287<=talkTime<=8000) | bill\_<bill |
| MR11 | talkTime\_=2\*talkTime, (286<=talkTime<=4000) | bill\_>=bill |
| MR12 | talkTime\_=talkTime/2, (286\*2<=talkTime<=8000) | bill\_<=bill |
| 4. Type=A，planFee=886，3000<=talkTime<=8000 & flow<=3000 | | |
| MR13 | talkTime\_=talkTime+1,(3000<=talkTime<=7999) | bill\_>bill |
| MR14 | talkTime\_=talkTime-1, (3001<=talkTime<=8000) | bill\_<bill |
| MR15 | talkTime\_=2\*talkTime, (3000<=talkTime<=4000) | bill\_>=bill |
| MR16 | talkTime\_=talkTime/2, (3000\*2<=talkTime<=8000) | bill\_<=bill |

表 5-6 中国联通计费程序蜕变关系描述（续表）

|  |  |  |
| --- | --- | --- |
| **蜕变关系**  **序号** | ***R*** | ***Rf*** |
| 5. Type=B，planFee=46，talkTime<=120 & 40<=flow<=1000 | | |
| MR17 | flow\_=flow+1,(40<=flow<=1000) | bill\_>bill |
| MR18 | flow\_=flow-1, (41<=flow<=1000) | bill\_<bill |
| MR19 | flow\_=2\*flow, (40<=flow<=500) | bill\_>=bill |
| MR20 | flow\_=flow/2, (40\*2<=flow<=1000) | bill\_<=bill |
| 6.Type=B，planFee=96，talkTime<=450 & 80<=flow<=1000 | | |
| MR21 | flow\_=flow+1,(80<=flow<=1000) | bill\_>bill |
| MR22 | flow\_=flow-1, (81<=flow<=1000) | bill\_<bill |
| MR23 | flow\_=2\*flow, (80<=flow<=500) | bill\_>=bill |
| MR24 | flow\_=flow/2, (80\*2<=flow<=1000) | bill\_<=bill |
| 7. Type=B，planFee=126，talkTime<=680 & 100<=flow<=1000 | | |
| MR25 | flow\_=flow+1,(100<=flow<=1000) | bill\_>bill |
| MR26 | flow\_=flow-1, (101<=flow<=1000) | bill\_<bill |
| MR27 | flow\_=2\*flow, (100<=flow<=500) | bill\_>=bill |
| MR28 | flow\_=flow/2, (100\*2<=flow<=1000) | bill\_<=bill |
| 8. Type=B，planFee=186，talkTime<=1180 & 150<=flow<=1000 | | |
| MR29 | flow\_=flow+1,(150<=flow<=1000) | bill\_>bill |
| MR30 | flow\_=flow-1, (151<=flow<=1000) | bill\_<bill |
| MR31 | flow\_=2\*flow, (150<=flow<=500) | bill\_>=bill |
| MR32 | flow\_=flow/2, (150\*2<=flow<=1000) | bill\_<=bill |

表 5-6 中国联通计费程序蜕变关系描述（续表）

|  |  |  |
| --- | --- | --- |
| **蜕变关系**  **序号** | ***R*** | ***Rf*** |
| 属性蜕变关系设计 | | |
| 9. Type=A，planFee=286，talkTime<=285 & flow<=900 | | |
| MR33 | talkTime\_=talkTime+1 | bill\_ = bill |
| 10. Type=A，planFee=286，talkTime<=286 & flow<=450 | | |
| MR34 | flow\_=2\*flow | bill\_ = bill |
| 11. Type=B，planFee=186，talkTime<=1180 & flow<=150 | | |
| MR35 | talkTime\_=talkTime-1 | bill\_ = bill |
| MR36 | flow\_=0.5\*flow | bill\_ = bill |

1. **停车计费程序**

停车计费程序是根据某停车场的计费标准实现的，程序根据司机的车辆类型（摩托车或汽车），汽车类型（跑车或轿车），停车日期（工作日或周末），折扣券和停车时间计算停车费用。程序采用Java实现。

1. **输入规格说明**

表 5-7 停车计费程序输入参数描述

|  |  |  |
| --- | --- | --- |
| 输入参数 | 参数类型 | 限制 |
| typeOfVehical | int | 仅为数字0,1 |
| typeOfCar | int | 仅为数字0,1 |
| dayOfWeek | int | 仅为数字0,1 |
| apd | double | 仅为0-24的实数 |
| discountCoupon | boolean |  |
| estimation | String | 仅为：“(0.0,2.0]”，“(2.0,4.0]”，“(4.0,24.0]”，null |

说明：在中，typeOfVehical代表车辆类型，0,1分别代表摩托车和汽车；typeOfCar代表汽车的类型，0,1分别代表跑车和轿车；dayOfWeek代表停车的日期，0,1分别代表工作日和周末；apd代表实际停车时间；discountCoupon表示是否使用优惠券；estimation代表司机预估停车时间，分成表中四个时间段，其中null表示司机未给出预估停车时间。

1. **输出规格说明**

调用程序的parkingFee接口得到double型的输出参数fee。输出结果根据输入不同分成三种情况：

1. 用户提供优惠券：当用户提供优惠券，可在原有的停车费用上享受五折优惠。
2. 用户选择预估停车时间：用户可以根据自身情况选择预估的停车时间，分成三个时间段，“不超过2小时”、“2小时到4小时”和“4小时到24小时”。如果停车的时间在预估的时间段之内，司机可以享受四折优惠，反之，司机需要在原有停车费之外多付20%的附加管理费。
3. 用户既没有优惠券也不提供预估停车时间：如果司机不提供优惠券，也不提供预估的停车时间，停车费用不享受折扣。

需要说明的是，用户不能够在提供优惠券的同时给出预估停车时间，只能选择一种优惠方式或者都不选择。

当输入符合程序规格说明时，根据停车时间，是否享受折扣，停车单价计算出最终的停车费用。停车单价计算方式如所示。

表 5-8 停车计费单价计算规则

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 停车时间(单位：小时) | 停车单价(单位：元) | | | | | |
| 工作日 | | | 周末 | | |
| 摩托车 | 跑车 | 轿车 | 摩托车 | 跑车 | 轿车 |
| (0.0,2.0] | 4.00 | 4.50 | 5.00 | 5.00 | 6.00 | 7.00 |
| (2.0,4.0] | 5.00 | 5.50 | 6.00 | 6.50 | 7.50 | 8.50 |
| (4.0,24.0] | 6.00 | 6.50 | 7.00 | 8.00 | 9.00 | 10.00 |

1. **蜕变关系描述**

在实验中，根据预估停车时间对程序的输入进行等价类划分，在每一个等价类中依据基于数据变异的蜕变关系获取技术生成8条蜕变关系，依据程序的属性设计1条蜕变关系。共9条蜕变关系如所示。

表 5-9 停车计费程序蜕变关系集

|  |  |  |
| --- | --- | --- |
| **蜕变关系**  **序号** | ***R*** | ***Rf*** |
| 1. estimation=“(0.0,2.0]” | | |
| MR1 | apd\_=apd+0.1(0.0<apd<=1.9) | fee\_>=fee |
| MR2 | apd\_=apd-0.1, (0.1<apd<=2.0) | fee\_<=fee |
| 2. estimation=“(2.0,4.0]” | | |
| MR3 | apd\_=apd+0.1(2.0<apd<=3.9) | fee\_>=fee |
| MR4 | apd\_=apd-0.1(2.1<apd<=4.0) | fee\_<=fee |
| 3. estimation=“(4.0,24.0]” | | |
| MR5 | apd\_=apd+0.1(4.0<apd<=23.9) | fee\_>=fee |
| MR6 | apd\_=apd-0.1(4.1<apd<=24) | fee\_<=fee |
| 4. estimation=null | | |
| MR7 | apd\_=apd+0.1(0.0<apd<=23.9) | fee\_>=fee |
| MR8 | apd\_=apd-0.1(0.1<apd<=24.0) | fee\_<=fee |
| 5. 属性蜕变关系 | | |
| MR9 | apd\_=ceil(apd),(0.0<apd<=24.0) | fee\_=fee |

说明：蜕变关系MR9中的ceil( )代表上取整。

1. 变异体与测试用例生成

用变异体生成工具MuJava为停车计费程序生成754个非等价变异体。

根据3.3节中的原始测试用例生成方法，解析停车计费程序的路径，并生成满足路径的原始测试用例，共生成原始测试用例144个。应用随机值法生成原始测试用例集规模分别为TC=144，TC=288，TC=720，TC=1440。

* 1. 实验结果分析

统计两组对比实验的结果，根据变异得分对实验结果进行分析，分析将围绕5.1节的三个问题展开讨论。

1. **基于符号执行的原始测试用例生成技术的可行性与适用性**

通过基于符号执行的原始测试用例生成技术，对5.3小节中的三个程序都能有效的生成高覆盖率的原始测试用例。航空行李托运计费程序、联通计费程序和停车计费程序分别生成34，32，144个原始测试用例。实验结果表明该技术可用于蜕变测试中原始测试用例的生成。

由于约束求解目前不支持非线性问题和复杂类型的求解，因此该方法在某些方面具有局限性。若程序中存在非线性的操作，需要对程序进行预处理,将非线性问题转成线性问题。

1. **基于符号执行的原始测试用例生成技术的有效性**

通过对三个实例进行对比实验，比较基于符号执行的原始测试用例生成方法与原始测试用例随机生成方法的故障检测能力。

1. **实例研究1：航空行李托运计费程序**

在航空行李托运计费程序中，设计36条蜕变关系，采用基于符号执行的方法生成34个原始测试用例，随机生成4组原始测试用例集，规模分别是TC=34，TC=68，TC=170，TC=340。分别用5组原始测试用例集和36条蜕变关系进行蜕变测试，其中所测试的变异体集相同，规模为56个，比较五组实验的变异得分。实验结果如所示。

在中，SC代表采用基于符号执行方法生成的原始测试用例集，R1n，R2n，R5n，R10n分别代表用随机值法生成1倍，2倍，5倍，10倍规模的原始测试用例集。

从总的变异得分来看，在原始测试用例集规模相同的情况下，基于符号执行的原始测试用例生成方法的变异得分为73.21%，而随机生成原始测试用例的方法的变异得分仅为41.07%。随机生成2倍原始测试用例情况下，变异得分依然没有很大的提高，当测试用例规模达到5倍时，变异得分达到71.43%，之后再增加原始测试用例规模，变异得分也没有变化。这说明了用随机值法生成原始测试用例时存在大量的冗余，而基于符号执行的原始测试用例生成方法能以更少的测试用例检测到更多的故障。

就MR1~MR36的测试结果而言，基于符号执行的原始测试用例生成方法的故障检测能力基本都高于随机值法，变异得分分布在14.29%~48.21%范围内。当用随机值法生成相等规模测试用例集进行蜕变测试的时候，变异得分分布在7.14%~28.57%。描述了用两种方法生成同等规模测试用例集的情况下，36条蜕变关系的变异得分。从表中可以看出，基于符号执行的测试用例生成方法的变异得分明显高于随机值法，变异得分提高比例范围在14.32%~200%。主要原因在于前者生成的原始测试用例集路径覆盖率高，并且通过约束求解得到的测试用例通常是路径的边界值，对于程序中的故障更加敏感。

1. **实例研究2：联通计费程序**

从中可以看出，基于符号执行的原始测试用例生成方法故障检测能力明显高于随机值法，当测试用例集规模为TC=32，前者的总变异得分为53.27%，后者的总变异得分仅为25.00%。增大随机测试用例集规模，总变异得分提升并不明显，随机生成320个原始测试用例进行蜕变测试，36条蜕变关系的总变异得分仅达到35.71%。通过对随机生成的测试用例集分析，发现虽然测试用例集已经覆盖程序中所有的路径，但是对于程序中比较微小的变异如变量自增1变异操作，则需要考虑边界值情况才能将变异体杀死。

从每一条蜕变关系的变异得分分析，基于符号执行的测试用例生成方法的故障检测能力相比随机值法有不同程度的提高。显示了在原始测试用例集规模为TC=32时，两种方法的变异得分结果。利用随机值法生成32个原始测试用例进行蜕变测试，其中有15条蜕变关系的变异得分为0，原因是随机生成的原始测试用例集中没有满足这15条蜕变关系蜕变域的测试用例。在这种情况下需要增加测试用例的数量以生成满足蜕变关系蜕变域的原始测试用例，从中看出，当测试用例集规模在TC=320时，在每条蜕变关系的蜕变域内都生成了原始测试用例，但是故障检测能力依然不如基于符号执行生成的测试用例。

1. **实例研究3：停车计费程序**

从的测试结果看来，针对停车计费程序，采用基于符号执行的方法生成144个测试用例，设计9条蜕变关系进行蜕变测试，共杀死33.55%的变异体；随机生成相同规模的测试用例，采用相同的蜕变关系集进行蜕变测试，9条蜕变关系总变异得分为27.72%。从总的变异得分看，在测试用例集规模相等的情况下，基于符号执行的方法的故障检测能力要高于随机值法，即使增大随机值法生成测试用例的规模，总变异得分的提升也不明显。

从的对比结果看，针对MR1~MR9，在测试用例集规模相同的情况下， 基于符号执行的原始测试用例生成方法故障检测能力比随机值法有不同程度的提升，提升的范围在16.54%~135.85%。

在本实例中，针对设计的9条蜕变关系，采用基于符号执行的方法生成的原始测试用例和随机生成的原始测试用例进行蜕变测试，总的变异得分都不是很高，原因可能是停车计费程序的结构较为复杂，在程序实现中用到了枚举等复杂的变量类型，生成的变异体较多，而设计的9条蜕变关系的结构过于简单，这种情况需要根据程序的属性设计更高效的蜕变关系。

表 5-10 航空行李托运计费程序变异得分

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **蜕变**  **关系** | **变异得分** | | | | |
| **SC** | **R1n** | **R2n** | **R5n** | **R10n** |
| **MR1** | 25.00% | 16.07% | 16.07% | 16.07% | 17.86% |
| **MR2** | 25.00% | 16.07% | 16.07% | 16.07% | 17.86% |
| **MR3** | 14.29% | 8.93% | 8.93% | 8.93% | 12.50% |
| **MR4** | 14.29% | 12.50% | 12.50% | 16.07% | 16.07% |
| **MR5** | 26.79% | 17.86% | 19.64% | 21.43% | 25.00% |
| **MR6** | 26.79% | 17.86% | 19.64% | 21.43% | 21.43% |
| **MR7** | 16.07% | 12.50% | 12.50% | 12.50% | 16.07% |
| **MR8** | 16.07% | 12.50% | 12.50% | 12.50% | 12.50% |
| **MR9** | 28.57% | 16.07% | 16.07% | 17.86% | 26.79% |
| **MR10** | 28.57% | 14.29% | 16.07% | 17.86% | 23.21% |
| **MR11** | 14.29% | 10.71% | 10.71% | 10.71% | 16.07% |
| **MR12** | 14.29% | 8.93% | 12.50% | 12.50% | 12.50% |
| **MR13** | 37.50% | 28.57% | 30.36% | 30.36% | 30.36% |
| **MR14** | 37.50% | 28.57% | 30.36% | 30.36% | 30.36% |
| **MR15** | 26.79% | 21.43% | 21.43% | 21.43% | 21.43% |
| **MR16** | 26.79% | 21.43% | 21.43% | 21.43% | 21.43% |
| **MR17** | 25.00% | 14.29% | 17.86% | 17.86% | 19.64% |
| **MR18** | 25.00% | 16.07% | 17.86% | 17.86% | 19.64% |
| **MR19** | 14.29% | 10.71% | 10.71% | 10.71% | 10.71% |
| **MR20** | 14.29% | 8.93% | 12.50% | 12.50% | 16.07% |
| **MR21** | 26.79% | 14.29% | 16.07% | 16.07% | 17.86% |
| **MR22** | 26.79% | 16.07% | 16.07% | 17.86% | 17.86% |
| **MR23** | 16.07% | 8.93% | 10.71% | 10.71% | 10.71% |
| **MR24** | 16.07% | 8.93% | 10.71% | 12.50% | 12.50% |
| **MR25** | 26.79% | 14.29% | 16.07% | 16.07% | 19.64% |
| **MR26** | 26.79% | 16.07% | 16.07% | 17.86% | 17.86% |
| **MR27** | 14.29% | 7.14% | 8.93% | 10.71% | 12.50% |
| **MR28** | 14.29% | 10.71% | 12.50% | 12.50% | 12.50% |
| **MR29** | 35.71% | 25.00% | 25.00% | 41.07% | 41.07% |
| **MR30** | 35.71% | 23.21% | 25.00% | 41.07% | 41.07% |
| **MR31** | 26.79% | 19.64% | 19.64% | 21.43% | 21.43% |
| **MR32** | 26.79% | 19.64% | 19.64% | 21.43% | 21.43% |
| **MR33** | 48.21% | 16.07% | 16.07% | 19.64% | 46.43% |
| **MR34** | 48.21% | 16.07% | 16.07% | 46.43% | 42.86% |
| **MR35** | 16.07% | 10.71% | 10.71% | 12.50% | 16.07% |
| **MR36** | 16.07% | 8.93% | 12.50% | 12.50% | 12.50% |
| **total** | 73.21% | 41.07% | 42.86% | 71.43% | 71.43% |

表 5-11 航空行李托运程序对比实验结果

|  |  |  |  |
| --- | --- | --- | --- |
| **蜕变**  **关系** | **变异得分** | | **提高比例** |
| **R1n** | **SC** |
| **MR1** | 16.07% | 25.00% | 55.57% |
| **MR2** | 16.07% | 25.00% | 55.57% |
| **MR3** | 8.93% | 14.29% | 60.02% |
| **MR4** | 12.50% | 14.29% | 14.32% |
| **MR5** | 17.86% | 26.79% | 50.00% |
| **MR6** | 17.86% | 26.79% | 50.00% |
| **MR7** | 12.50% | 16.07% | 28.56% |
| **MR8** | 12.50% | 16.07% | 28.56% |
| **MR9** | 16.07% | 28.57% | 77.78% |
| **MR10** | 14.29% | 28.57% | 99.93% |
| **MR11** | 10.71% | 14.29% | 33.43% |
| **MR12** | 8.93% | 14.29% | 60.02% |
| **MR13** | 28.57% | 37.50% | 31.26% |
| **MR14** | 28.57% | 37.50% | 31.26% |
| **MR15** | 21.43% | 26.79% | 25.01% |
| **MR16** | 21.43% | 26.79% | 25.01% |
| **MR17** | 14.29% | 25.00% | 74.95% |
| **MR18** | 16.07% | 25.00% | 55.57% |
| **MR19** | 10.71% | 14.29% | 33.43% |
| **MR20** | 8.93% | 14.29% | 60.02% |
| **MR21** | 14.29% | 26.79% | 87.47% |
| **MR22** | 16.07% | 26.79% | 66.71% |
| **MR23** | 8.93% | 16.07% | 79.96% |
| **MR24** | 8.93% | 16.07% | 79.96% |
| **MR25** | 14.29% | 26.79% | 87.47% |
| **MR26** | 16.07% | 26.79% | 66.71% |
| **MR27** | 7.14% | 14.29% | 100.14% |
| **MR28** | 10.71% | 14.29% | 33.43% |
| **MR29** | 25.00% | 35.71% | 42.84% |
| **MR30** | 23.21% | 35.71% | 53.86% |
| **MR31** | 19.64% | 26.79% | 36.41% |
| **MR32** | 19.64% | 26.79% | 36.41% |
| **MR33** | 16.07% | 48.21% | 200.00% |
| **MR34** | 16.07% | 48.21% | 200.00% |
| **MR35** | 10.71% | 16.07% | 50.05% |
| **MR36** | 8.93% | 16.07% | 79.96% |

表 5-12 联通计费程序实验结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **蜕变**  **关系** | **变异得分** | | | | |
| **SC** | **R1n** | **R2n** | **R5n** | **R10n** |
| **MR1** | 19.64% | 0.00% | 15.18% | 15.18% | 15.18% |
| **MR2** | 19.64% | 0.00% | 15.18% | 15.18% | 16.07% |
| **MR3** | 8.04% | 0.00% | 4.46% | 5.36% | 5.36% |
| **MR4** | 8.04% | 0.00% | 6.25% | 6.25% | 6.25% |
| **MR5** | 18.75% | 15.18% | 15.18% | 15.18% | 15.18% |
| **MR6** | 18.75% | 15.18% | 15.18% | 15.18% | 15.18% |
| **MR7** | 8.04% | 5.36% | 6.25% | 6.25% | 7.14% |
| **MR8** | 8.04% | 7.14% | 7.14% | 7.14% | 7.14% |
| **MR9** | 18.75% | 15.18% | 15.18% | 15.18% | 15.18% |
| **MR10** | 18.75% | 15.18% | 15.18% | 15.18% | 15.18% |
| **MR11** | 8.04% | 6.25% | 6.25% | 7.14% | 7.14% |
| **MR12** | 8.04% | 7.14% | 7.14% | 7.14% | 7.14% |
| **MR13** | 18.75% | 15.18% | 15.18% | 15.18% | 15.18% |
| **MR14** | 18.75% | 15.18% | 15.18% | 15.18% | 15.18% |
| **MR15** | 8.04% | 0.00% | 5.36% | 5.36% | 5.36% |
| **MR16** | 8.04% | 0.00% | 4.46% | 5.36% | 5.36% |
| **MR17** | 17.86% | 0.00% | 0.00% | 9.82% | 15.18% |
| **MR18** | 17.86% | 0.00% | 0.00% | 8.93% | 14.29% |
| **MR19** | 8.04% | 0.00% | 0.00% | 4.46% | 5.36% |
| **MR20** | 8.04% | 0.00% | 0.00% | 6.25% | 6.25% |
| **MR21** | 17.86% | 9.82% | 14.29% | 14.29% | 15.18% |
| **MR22** | 17.86% | 8.93% | 14.29% | 14.29% | 15.18% |
| **MR23** | 8.04% | 3.57% | 4.46% | 6.25% | 6.25% |
| **MR24** | 8.04% | 5.36% | 6.25% | 7.14% | 7.14% |
| **MR25** | 13.39% | 9.82% | 9.82% | 9.82% | 9.82% |
| **MR26** | 13.39% | 8.93% | 9.82% | 9.82% | 10.71% |
| **MR27** | 8.04% | 0.00% | 5.36% | 6.25% | 6.25% |
| **MR28** | 8.04% | 6.25% | 7.14% | 7.14% | 7.14% |
| **MR29** | 13.39% | 8.93% | 9.82% | 9.82% | 9.82% |
| **MR30** | 13.39% | 9.82% | 9.82% | 9.82% | 10.71% |
| **MR31** | 8.04% | 4.46% | 5.36% | 6.25% | 6.25% |
| **MR32** | 8.04% | 7.14% | 7.14% | 7.14% | 7.14% |
| **MR33** | 6.25% | 0.00% | 0.00% | 0.00% | 6.25% |
| **MR34** | 4.46% | 0.00% | 0.00% | 0.00% | 11.61% |
| **MR35** | 15.18% | 0.00% | 6.25% | 6.25% | 6.25% |
| **MR36** | 25.00% | 0.00% | 16.07% | 16.07% | 16.07% |
| **total** | 53.57% | 25.00% | 35.71% | 35.71% | 35.71% |

表 5-13 联通计费程序对比实验结果

|  |  |  |  |
| --- | --- | --- | --- |
| **蜕变**  **关系** | **变异得分** | | **提高比例** |
| **R1n** | **SC** |
| **MR1** | 0.00% | 19.64% | N/A |
| **MR2** | 0.00% | 19.64% | N/A |
| **MR3** | 0.00% | 8.04% | N/A |
| **MR4** | 0.00% | 8.04% | N/A |
| **MR5** | 15.18% | 18.75% | 23.52% |
| **MR6** | 15.18% | 18.75% | 23.52% |
| **MR7** | 5.36% | 8.04% | 50.00% |
| **MR8** | 7.14% | 8.04% | 12.61% |
| **MR9** | 15.18% | 18.75% | 23.52% |
| **MR10** | 15.18% | 18.75% | 23.52% |
| **MR11** | 6.25% | 8.04% | 28.64% |
| **MR12** | 7.14% | 8.04% | 12.61% |
| **MR13** | 15.18% | 18.75% | 23.52% |
| **MR14** | 15.18% | 18.75% | 23.52% |
| **MR15** | 0.00% | 8.04% | N/A |
| **MR16** | 0.00% | 8.04% | N/A |
| **MR17** | 0.00% | 17.86% | N/A |
| **MR18** | 0.00% | 17.86% | N/A |
| **MR19** | 0.00% | 8.04% | N/A |
| **MR20** | 0.00% | 8.04% | N/A |
| **MR21** | 9.82% | 17.86% | 81.87% |
| **MR22** | 8.93% | 17.86% | 100.00% |
| **MR23** | 3.57% | 8.04% | 125.21% |
| **MR24** | 5.36% | 8.04% | 50.00% |
| **MR25** | 9.82% | 13.39% | 36.35% |
| **MR26** | 8.93% | 13.39% | 49.94% |
| **MR27** | 0.00% | 8.04% | N/A |
| **MR28** | 6.25% | 8.04% | 28.64% |
| **MR29** | 8.93% | 13.39% | 49.94% |
| **MR30** | 9.82% | 13.39% | 36.35% |
| **MR31** | 4.46% | 8.04% | 80.27% |
| **MR32** | 7.14% | 8.04% | 12.61% |
| **MR33** | 0.00% | 6.25% | N/A |
| **MR34** | 0.00% | 4.46% | N/A |
| **MR35** | 0.00% | 15.18% | N/A |
| **MR36** | 0.00% | 25.00% | N/A |

表 5-14 停车计费程序实验结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **蜕变**  **关系** | **变异得分** | | | | |
| **SC** | **R1n** | **R2n** | **R5n** | **R10n** |
| **MR1** | 18.83% | 0.00% | 11.27% | 16.71% | 17.11% |
| **MR2** | 19.50% | 0.00% | 11.27% | 16.71% | 17.11% |
| **MR3** | 16.58% | 7.03% | 9.68% | 12.33% | 12.47% |
| **MR4** | 15.12% | 9.68% | 10.61% | 12.20% | 12.47% |
| **MR5** | 16.18% | 13.00% | 14.59% | 16.18% | 16.18% |
| **MR6** | 15.92% | 13.66% | 14.46% | 15.38% | 15.38% |
| **MR7** | 30.90% | 21.75% | 23.61% | 26.13% | 26.53% |
| **MR8** | 29.71% | 21.09% | 22.94% | 24.54% | 24.54% |
| **MR9** | 30.37% | 24.27% | 24.80% | 25.60% | 25.73% |
| **total** | 33.55% | 27.72% | 28.65% | 31.96% | 32.63% |

表 5-15 停车计费程序对比实验结果

|  |  |  |  |
| --- | --- | --- | --- |
| **蜕变**  **关系** | **变异得分** | | **提高比例** |
| **R1n** | **SC** |
| **MR1** | 0.00% | 18.83% | N/A |
| **MR2** | 0.00% | 19.50% | N/A |
| **MR3** | 7.03% | 16.58% | 135.85% |
| **MR4** | 9.68% | 15.12% | 56.20% |
| **MR5** | 13.00% | 16.18% | 24.46% |
| **MR6** | 13.66% | 15.92% | 16.54% |
| **MR7** | 21.75% | 30.90% | 42.07% |
| **MR8** | 21.09% | 29.71% | 40.87% |
| **MR9** | 24.27% | 30.37% | 25.13% |

1. **基于路径距离的原始测试用例优先级排序的有效性**

采用基于符号执行的原始测试用例生成方法，对航空行李托运计费程序、联通计费程序和停车计费程序分别生成34，32，144个原始测试用例。为了验证3.4节中原始测试用例优先级排序的有效性，分别从按优先级排序的原始测试用例集和未排序的原始测试用例集中选取相同比例的原始测试用例进行蜕变测试，比较两组测试用例的变异得分。实验结果如、图 5-2和图 5-3所示。

从、和图 5-3中的趋势上看，从按优先级排序的原始测试用例集中选取的原始测试用例随着选取比例的增大，变异得分的提高得更快，也更快的趋于稳定。在航空行李托运程序实验中，从优先级排序的原始测试用例集中选取40%的原始测试用例进行蜕变测试，变异得分已经达到70%以上，而从未排序的原始测试用例集中需要选取80%的原始测试用例进行蜕变测试，才能得到相同的变异得分。在联通计费程序和停车计费程序实验中，当选取比例相同的情况下，从按优先级排序的原始测试用例集中选取的原始测试用例故障检测能力明显高于从未按优先级排序的原始测试用例集中选取的原始测试用例，这说明基于路径距离的原始测试用例优先级排序方法的有效性。

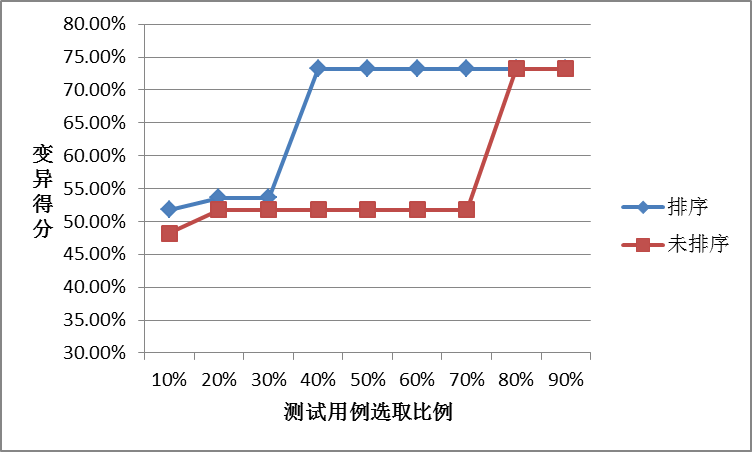


图 5-1 航空行李托运计费程序:原始测试用例选择对比实验结果

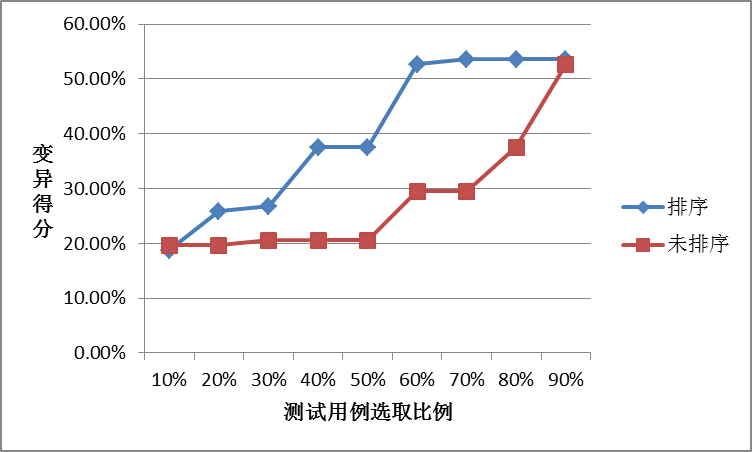


图 5-2 联通计费程序:原始测试用例选择对比实验结果

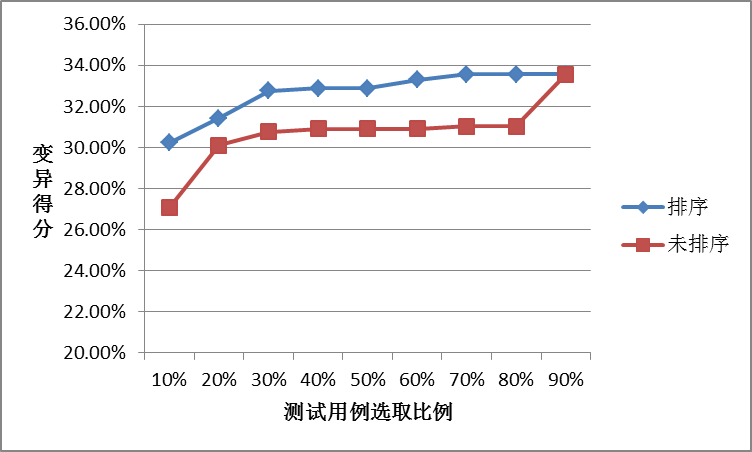


图 5-3 停车计费程序:原始测试用例选择对比实验结果

* 1. 本章小结

通过三个实例研究对基于符号执行的原始测试用例生成方法进行评估。实验结果表明，在同等测试用例集规模情况下，选用相同的蜕变关系集，本文提出的方法生成的原始测试用例集故障检测能力明显高于随机生成的原始测试用例集。

从按优先级排序的原始测试用例集中和未按优先级排序的原始测试用例集中选取相同比例的原始测试用例，进行蜕变测试，对比两者的变异得分。实验结果表明，从按优先级排序的原始测试用例集中选取的原始测试用例变异得分高于未按优先级排序的原始测试，从而验证了基于路径距离的原始测试用例优先级排序的有效性。

1. 工作总结与展望

蜕变测试是一种有效的缓解测试预期问题的测试技术。在蜕变测试过程中，原始测试用例的生成作为重要的一个环节，直接影响蜕变测试的故障检测能力。本文从待测源程序的路径覆盖角度出发，探索生成故障检测能力高的原始测试用例，进而改进蜕变测试的性能。本文结合符号执行和约束求解技术，提出了一种原始测试用例生成方法，根据源程序的路径约束信息获取高效的原始测试用例；根据符号执行获取的路径距离信息，设计优先级排序算法，对生成的原始测试用例按优先级进行排序；开发了基于符号执行的原始测试用例生成技术的支持工具，并集成到课题组的蜕变测试工具MT4WS上；选用3个典型的多分支程序进行实例研究，通过对比实验，验证所提出方法的有效性。

本文取得的主要成果如下：

* 提出了一种基于符号执行的原始测试用例生成技术，提高了蜕变测试的故障检测能力。
* 提出一种基于路径距离的原始测试用例优先级排序方法
* 开发了基于符号执行的原始测试用例生成技术的支持工具，进一步提升了蜕变测试的自动化程度。
* 采用实例研究的方式，验证了基于符号执行的原始测试用例生成技术可行性与有效性，并验证了基于路径距离的原始测试用例优先级排序方法的有效性。

工作不足及未来展望：

* 进一步研究基于符号执行的原始测试用例生成技术，扩展实例研究验证该方法的有效性。
* 进一步研究基于路径距离的原始测试用例优先级排序方法，充分利用获取的路径信息对原始测试用例进行排序。

参考文献

1. Weyuker E J. On testing non-testable programs [J]. The Computer Journal, 1982, 25(4): 465-470.
2. Harman M, McMinn P, Shahbaz M, et al. A comprehensive survey of trends in oracles for software testing, CS-13-01 [R]. Department of Computer Science, University of Sheffield, 2013.
3. Hierons R M, Oracles for distributed testing [J]. IEEE Transactions on Software Engineering, 2012, 38 (3): 629–641.
4. Chen T Y, Cheung S C, Yiu S M. Metamorphic testing: a new approach for generating next test cases, HKUST-CS98-01 [R]. Department of Computer Science, Hong Kong University of Science and Technology, 1998.
5. Chen T Y, Tse T H, Zhou Z Q. Fault-based testing without the need of oracles [J]. Information and Software Technology, 2003, 45 (1): 1–9.
6. Sun C, Wang G, Wen Q, et al. MT4WS: an automated metamorphic testing system for Web services [J]. International Journal of High Performance Computing and Networking, 2016, 9(1/2): 104-115.
7. Wang R, Ben K. Classification of metamorphic relations and its application [J]. American Journal of Engineering and Technology Research, 2011, 11(12): 1664-1668.
8. Chen T Y, Kuo F C, Liu Ying, et al. Metamorphic testing and testing with special values [C]//Proceeding of the 5th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2004), IEEE Computer Society, Beijing, China, 2004: 128-134.
9. 吴鹏, 施小纯, 唐江峻, 等. 关于蜕变测试和特殊用例测试的实例研究 [J]. 软件学报, 2005, 16(7): 1210-1220.
10. Guderlei R, Mayer J. Statistical metamorphic testing programs with random output by means of statistical hypothesis tests and metamorphic testing [C]//Proceedings of the 7th International Conference on Quality Software (QSIC 2007), IEEE Computer Society, Portland, Oregon, USA, 2007: 404-409.
11. 董国伟, 聂长海, 徐宝文. 基于程序路径分析的有效蜕变测试 [J]. 计算机学报, 2009, 32(5): 1002-1013.
12. Wu P. Iterative metamorphic testing [C]//Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), IEEE Computer Society, Edinburgh, Scotland, UK, 2005: 19-24.
13. Dong G, Nie C, Xu B, et al. An effective iterative metamorphic testing algorithm based on program path analysis [C]//Proceedings of the 7th International Conference on Quality Software (QSIC 2007), IEEE Computer Society, Portland, Oregon, USA, 2007: 292-297.
14. Chen T Y, Huang D H, Tse T H, et al. Case studies on the selection of useful relations in metamorphic testing [C]//Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004), 2004: 569-583.
15. Mayer J, Guderlei R. An empirical study on the selection of good metamorphic relations [C]//Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), IEEE Computer Society, Chicago, Illinois, USA, 2006,1: 475-484.
16. Asrafi M, Liu H, Kuo F C. On testing effectiveness of metamorphic relations: a case study [C]//Proceedings of the Fifth International Conference on Secure Software Integration and Reliability Improvement (SSIRI 2011), IEEE Computer Society, Jeju Island, Korea, 2011: 147-156.
17. 王瑢, 贲可荣. 蜕变关系构造基本准则与策略研究[J]. 计算机科学, 2012, 39(1): 115-119.
18. Liu H, Liu X, Chen T Y. A new method for constructing metamorphic relations [C]//Proceedings of the 12th International Conference on Quality Software (QSIC 2012), IEEE Computer Society, Xi'an, Shaanxi, China, 2012: 59-68.
19. Gotlieb A, Botella B. Automated metamorphic testing [C]//Proceedings of the 27th IEEE Annual International Computer Software and Applications Conference (COMPSAC 2003), IEEE Computer Society, Dallas, TX, USA, 2003: 34-40.
20. Segura S, Hierons R M, Benavides D, et al. Automated test data generation on the analyses of feature models: a metamorphic testing approach [C]//Proceedings of the Third International Conference on Software Testing Verification and Validation (ICST 2010), IEEE Computer Society, Paris, France, 2010: 35-44.
21. Murphy C, Shen K, Kaiser G. Automatic system testing of programs without test oracles [C]//Proceedings of the 8th International Symposium on Software Testing and Analysis (ISSTA 2009), Association for Computing Machinery, Chicago, IL, USA, 2009: 189-200.
22. Murphy C, Shen K, Kaiser G. Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles [C]//Proceedings of 2009 International Conference on Software Testing Verification and Validation (ICST 2009), IEEE Computer Society, Denver, Colorado, USA, 2009: 436-445.
23. Sun C, Wang G, Mu B, et al. A metamorphic relation-based approach to testing web services without oracles [J]. International Journal of Web Services Research, 2012, 9(1): 51-73.
24. Sun C A, Liu Y, Wang Z, et al. μMT: a data mutation directed metamorphic relation acquisition methodology [C]//Proceedings of the 1st International Workshop on Metamorphic Testing (MET 2016), ACM, Austin, Texas, USA, 2016: 12-18.
25. 林惠民, 张文辉. 模型检测:理论、方法与应用 [J]. 电子学报, 2002, 30(1):1907-1912.
26. Clarke E M. Model checking and abstraction [J]. ACM Transactions on Programming Languages & Systems, 1994, 16(5):1512-1542.
27. Havelund K, Pressburger T. Model Checking Java Programs Using Java PathFinder [J]. International Journal on Software Tools for Technology Transfer, 1999, 2(4):366-381.
28. King J C. A New Approach to Program Testing [J]. ACM Sigplan Notices, 1974, 10(6):278-290.
29. Clarke L A. A System to Generate Test Data and Symbolically Execute Programs [J]. Software Engineering IEEE Transactions on, 1976, 2(3):215-222.
30. <http://www.choco-solver.org/>
31. Xie X, Ho J W K, Murphy C, et al. Testing and validating machine learning classifiers by metamorphic testing [J]. Journal of Systems and Software, 2011, 84 (4), 544–558.
32. Segura S, Hierons R M, Benavides D, et al. Automated metamorphic testing on the analyses of feature models [J]. Information and Software Technology, 2011, 53 (3), 245–258.
33. Murphy C, Raunak M, King A, et al. On effective testing of health care simulation software [C]//Proceedings of the 3rd Workshop on Software Engineering in Health Care (SEHC 2011), Association for Computing Machinery, Honolulu, HI, USA, 2011: 40–47.
34. Poon P L, Kuo F C, Liu H, et al. How can non-technical end users effectively test their spreadsheets? [J]. Information Technology and People, 2014, 27 (4): 24-35.
35. 路晓丽, 董云卫. 面向服务软件的蜕变测试方法 [J]. 计算机应用, 2011, 31(7): 1756-1758.
36. 姚奕, 黄松, 稽孟雨. 面向整数错误检测的蜕变测试方法研究 [J]. 计算机工程与科学, 2012, 34(4): 52-56.
37. 黄松, 丁瑞浩, 李辉, 等. 坡度坡向量算程序蜕变测试方法 [J]. 计算机应用, 2013, 33(6): 1657-1661.
38. 林仁超, 刘晓明, 黄松, 等. 空间分析程序的蜕变测试框架 [J]. 攀枝花学院学报, 2013, 30(2): 113-119.
39. 张晶, 胡学钢, 张斌. 基于蜕变关系的聚类程序测试方法 [J]. 电子测量与仪器学报, 2011, 25(8): 688-694.
40. Murphy C, Shen K, Kaiser G. Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles [C]//Proceedings of the 2nd International Conference on Software Testing Verification and Validation (ICST 2009), IEEE Computer Society, Denver, Colorado, USA, 2009: 436-445.
41. Xie X, Wong W E, Chen T Y, et al. Metamorphic slice: an application in spectrumbased fault localization [J]. Information and Software Technology, 2013, 55 (5): 866–879.
42. Liu H, Yusuf I I, Schmidt H W, et al. Metamorphic fault tolerance: An automated and systematic methodology for fault tolerance in the absence of test oracle [C]//Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014), ACM Press, Hyderabad, India, 2014: 420–423.
43. 董国伟, 徐宝文, 陈林, 等. 使用复合蜕变关系进行软件测试的实例研究 (英文) [J]. Journal of Southeast University (English Edition), 2008, 24(4): 437-443.
44. 林仁超, 刘晓明, 黄松, 等. 基于复合函数的蜕变关系构造方法 [J]. 指挥信息系统与技术, 2013, 4(1): 66-69.
45. Hui Z, Huang S. A Formal Model for Metamorphic Relation Decomposition [C]//Proceedings of Fourth World Congress on Software Engineering (WCSE 2013), IEEE Computer Society, 2013: 64-48.
46. Kanewala U, Bieman J M. Using machine learning techniques to detect metamorphic relations for programs without test oracles [C]//Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE 2013), IEEE Computer Society, Pasadena, CA, USA, 2013: 1-10.
47. Sun C, Liu Y, Wang Z, Wen Q, Wu P, Chen T Y. An Empirical Study on Iterative Metamorphic Testing for Web Services [J]. Information and Software Technology. (已投稿)
48. Anand S, Godefroid P, Tillmann N. Demand-Driven Compositional Symbolic Execution.[M]// Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2008:367-381.
49. Palante G. Compositional dynamic test generation [J]. Acm Sigplan Notices, 2007, 42(1):47-54.
50. Rner N, Tillmann N, Voronkov A. Path Feasibility Analysis for String-Manipulating Programs [C]//Proceedings of the 15th International Conference on TOOLS & Algorithms for the Construction & Analysis of Systems (TACAS 2009), York, UK, 2009:307-321.
51. Khurshid S, Suen Y L. Generalizing symbolic execution to library classes.[J]. Acm Sigsoft Software Engineering Notes, 2005, 31(1):103-110.
52. Veanes M, Halleux P D, Tillmann N. Rex: Symbolic Regular Expression Explorer[C]//Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST 2010), IEEE Computer Society, Paris, France, 2010:498-507.
53. Ma K K, Phang K Y, Foster J S, et al. Directed Symbolic Execution[C]// Proceedings of the 18th Static Analysis International Symposium (SAS 2011), Venice, Italy, 2011:614 - 617.
54. Yang G, Person S, Rungta N, et al. Directed Incremental Symbolic Execution [J]. Acm Transactions on Software Engineering & Methodology, 2011, 46(6):504-515.
55. Saxena P, Poosankam P, Mccamant S, et al. Loop-extended symbolic execution on binary programs[C]//Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA 2009), Chicago, USA, 2009:225-236.
56. Chipounov V, Kuznetsov V, Candea G. S2E: a platform for in-vivo multi-path analysis of software systems[J]. Computer Architecture News, 2011, 39(1):265-278.

作者简历及在学研究成果

1. 作者入学前简历

|  |  |  |
| --- | --- | --- |
| 起止年月 | 学习或工作单位 | 备注 |
| 2010年9月至2014年6月 | 在北京科技大学自动化专业攻读 |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. 在学期间从事的科研工作

国家自然科学基金（面上项目）“面向SOA 软件的蜕变测试技术研究”（项目编号：61370061），项目主要参与者。课题时间：2014.01.01~2017.12.31

1. 在学期间所获的科研奖励

2016年10月 获研究生国家奖学金。

1. 在学期间发表的论文

第二作者. μMT: a data mutation directed metamorphic relation acquisition methodology [C]//Proceedings of the 1st International Workshop on Metamorphic Testing (MET 2016), ACM, Austin, Texas, USA, 2016: 12-18.(已发表，A类国际会议)

第二作者. An Empirical Study on Iterative Metamorphic Testing for Web Services [J]. Information and Software Technology. (返修中，SCI期刊)

独创性说明

本人郑重声明：所呈交的论文是我个人在导师指导下进行的研究工作及取得研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得北京科技大学或其他教育机构的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

签名：\_\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_\_\_

关于论文使用授权的说明

本人完全了解北京科技大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

**（保密的论文在解密后应遵循此规定）**

签名：\_\_\_\_\_\_\_\_\_\_\_ 导师签名：\_\_\_\_\_\_\_\_\_\_\_ 日期：\_\_\_\_\_\_\_\_\_\_\_\_

学位论文数据集

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **关键词\*** | **密级\*** | **中图分类号\*** | **UDC** | **论文资助** |
| 蜕变测试，原始测试用例，符号执行，约束求解 | 公开 | TP311 | 004.41 | 国家自然科学基金（面上项目）“面向SOA 软件的蜕变测试技术研究” |
| **学位授予单位名称\*** | | **学位授予单位代码\*** | **学位类别\*** | **学位级别\*** |
| 北京科技大学 | | 10008 | 工学 | 硕士 |
| **论文题名\*** | | **并列题名** | | **论文语种\*** |
| 基于符号执行的蜕变测试中原始测试用例生成技术与支持工具研究 | |  | | 中文 |
| **作者姓名\*** | 刘益强 | | **学号\*** | S20140931 |
| **培养单位名称\*** | | **培养单位代码\*** | **培养单位地址** | **邮编** |
| 北京科技大学 | | 10008 | 北京市海淀区学院路30号 | 100083 |
| **学科专业\*** | | **研究方向\*** | **学制\*** | **学位授予年\*** |
| 软件工程 | | 软件测试 | 2.5年 | 2017 |
| **论文提交日期\*** | 2016.12.22 | | | |
| **导师姓名\*** | 孙昌爱 | | **职称\*** | 教授 |
| **评阅人** | **答辩委员会主席\*** | | **答辩委员会成员** | |
|  | 张晓彤 | | 陈红松  姚宣霞 | |
| **电子版论文提交格式** 文本（ ） 图像（ ） 视频（ ） 音频（ ） 多媒体（ ） 其他（ ） **推荐格式：**application/msword；application/pdf | | | | |
| **电子版论文出版（发布）者** | | **电子版论文出版（发布）地** | | **权限声明** |
|  | |  | |  |
| **论文总页数\*** | 65 | | | |
| 共33项，其中带\*为必填数据，为22项。 | | | | |