基于子路径扩展的不可达路径检测方法

王红阳[1],姜淑娟[1],王兴亚[1],鞠小林[1,2],张艳梅[1]
(1.中国矿业大学计算机科学与技术学院,江苏徐州 221116; 2.南通大学计算机科学与技术学院,江苏南通 226019)

摘 要: 不可达路径是造成软件测试资源消耗的一个重要方面.在路径测试之前,检测程序中的不可达路径可以有效节约软件测试资源.提出了一种基于子路径扩展的不可达路径静态检测方法.该方法首先生成程序的子路径集,将路径的可达性问题转换为不等式组的求解问题.使用约束求解器判断子路径的可达性,可以分为:可达子路径,不可达子路径和无法判定三个部分,并对后面两部分的子路径扩展出的路径做二次可达性检测,最终获得程序中所有路径的可达性信息.可达性检测工作主要在子路径集上进行,因此有效地解决了路径爆炸问题.实验结果表明本文方法可以准确有效地检测出程序中的不可达路径.

关键词: 软件测试; 子路径扩展; 不可达路径检测; 约束求解
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2015)08-1555-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2015.08.013

An Approach for Detecting Infeasible Paths Based on Sub-Path Expansion

WANG Hong-yang[1], JIANG Shu-juan[1], WANG Xing-ya[1], JU Xiao-lin[1,2], ZHANG Yan-mei[1]
(1. *School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu* 221116, *China*;
2. *School of Computer Science and Technology, Nantong University, Nantong, Jiangsu* 226019, *China*)

Abstract: Infeasible paths are one of the most important parts to cost the test resources. Before path testing, infeasible paths detecting in a program can save test resources efficiently. This paper presents a static method to detect infeasible paths which is based on sub-path expansion. First, the proposed method generates sub-path set, and the feasibility issues will be converted into inequalities. Second, the constraint solver is used to solve the inequalities, and then we can distinguish the sub-paths into three parts: one part is infeasible sub-paths, the second part is feasible sub-paths, and the third part is undetermined. The paths that are expanded from the latter two parts will be tested again to determine their feasibility. Eventually, the feasibility of all the paths is detected. Most of the detecting work is on sub-path set, so our method makes an effective solution to the path-explosion problem. Experimental results show that the proposed method can detect infeasible paths more accurately and effectively.

Key words: software testing; sub-path expansion; infeasible path detection; constraint solving

1 引言

软件测试是保证软件质量的重要手段.测试工作中,如果测试人员期望覆盖的路径是一条不可达路径,那么针对该路径的测试用例生成工作会耗费大量的时间与精力,却达不到预期的测试目标,造成测试资源的浪费.人们需要在测试工作之前,识别出不可达路径以避免资源浪费.可见,针对不可达路径检测技术的研究具有重要意义.

针对程序中不可达路径的检测问题,本文提出一种基于子路径扩展的不可达路径静态检测方法 SPE-IPD (Sub-Path Expansion Based Infeasible Path Detecting).该方法首先生成子路径集,然后抽取子路径中的路径条件,构造不等式组并通过约束求解器求解,根据求解结果判断子路径是否可达,最终扩展为路径.如果子路径不可达,则扩展出的路径也不可达,如果子路径可达或是无法判定,则需对扩展出的路径做二次检测,最终得到所有路径的可达性信息.本文方法主要检测工作在子路径集上进行,对于结构复杂的程序,子路径数目远远少于路径数目,因此有效的解决了路径爆炸问题.

收稿日期:2014-09-23;修回日期:2014-12-15;责任编辑:马兰英
基金项目:国家自然科学基金(No.61340037,No.61202006);中央高校基本科研业务费专项资金(No.2013NB17);江苏省青蓝工程;江苏省 333 高层次人才培养工程项目;江苏省研究生培养创新工程(No.CXZZ12-0935);南通市应用研究计划(No.BK2014055)

2 相关定义

定义 1 路径 路径是程序中顺序执行的一组语句序列.

定义 2 子路径 子路径是路径的一个片段.区别是,路径的起点为控制流图的入口节点,终点为控制流图的出口节点;子路径起点为控制流图的入口节点或分支节点,终点为控制流图的出口节点.

定义 3 相关变量 程序中分支语句谓词 p 一般可以表示为两种形式,v_1 op_1 v_2或 v_3 op_2 c,其中 v_1,v_2与 v_3 是谓词中的变量,c 是常量,op_1与 op_2代表判定符号.则称 v_1,v_2与 v_3是谓词 p 的相关变量.

定义 4 变量定值点 程序中变量 v 的定值语句节点称为变量定值点.如果变量 v 是谓词 p 的相关变量,则称为相关变量定值点,记为 $dp(v)$.

3 本文方法 SPE-IPD

提出一种基于子路径扩展的不可达路径静态检测方法 SPE-IPD,其架构图如图 1 所示.

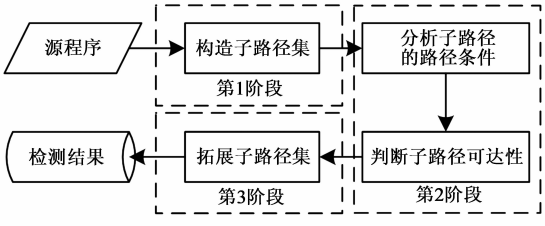


图1 SPE-IPD方法框架

SPE-IPD 可以分为 3 个阶段:第 1 阶段,构造子路径集.通过对无环的控制流图进行递归遍历,构造子路径集;第 2 阶段,判断子路径可达性.对每一条子路径,分析路径条件,根据约束求解器的求解结果判断子路径的可达性;第 3 阶段,扩展子路径集,完善不可达路径检测结果,最终得到程序中所有路径的可达性信息.

3.1 构造子路径集

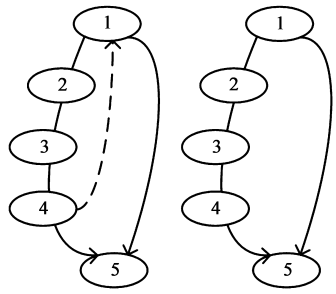
本阶段首先对程序的控制流图进行去环,通过添加辅助的边或删掉无用的边,消去循环中的回边,所得到的控制流图是一个有向无环图.然后,通过对无环的控制流图进行递归遍历而构造子路径集.

控制流图去环目的是约简程序中存在的循环.本文方法将循环的执行过程视为两种情况.

(1)循环内部代码执行一遍.如果循环内包含条件分支,则条件的真假分支各算一条执行路径.

(2)循环内部代码不执行,由循环入口直接跳转到循环出口处.

图 2 为控制流图去环示例,图 2(a)为去环之前的控制流图,图 2 中 1-2-3-4-1 是一个循环,形成了一个环,在图 2(a)中,删除 4-1 的虚线回边,得到图 2(b),即完成控制流图的去环工作.



(a) 去环前控制流图;(b) 去环后控制流图

图2 控制流图去环实例

去环后的控制流图包含以下两条路径.

(1){1,2,3,4,5},即循环内代码执行一遍.

(2){1,5},即循环内代码不执行,由循环入口直接跳转到循环出口处.

SPE-IPD 完成控制流图去环工作后,根据子路径集构造算法构造子路径集.

算法 1 子路径集构造算法

```
algorithm:sub-path set generation
input:startLine                          //待分析程序的起始行
output:subPathSet
function subPathGeneration(startLine)    //递归调用
begin
1.  sp←sp∪startLine
2.  if startLine has successors then
3.     foreach line in startLine.successors
4.        subPathGeneration(line)        //递归调用
5.     endfor
6.  else
7.     subPathSet←subPathSet∪sp
8.     sp←Ø
9.  endif
end
```

SPE-IPD 在构造子路径集时采用了递归的算法.由于在构造子路径集的过程中去除了控制流图中的环,因此递归算法可以达到终止的条件.终止条件是某一个节点没有任何的后继节点,即第 2 行判定为假,此类节点是控制流图的出口节点.到达递归终止条件时,sp 为一条子路径,将其并入子路径集 subPathSet 中后,清空 sp 中存储的语句节点(第 8 行),根据递归方法的调用堆栈,返回上层调用方法继续查找子路径(第 4 行).

3.2 判断子路径可达性

描述本阶段工作之前,首先解释判断子路径可达性的原理.引起子路径不可达的原因有两种情况.

(1)两个分支语句的特定分支条件有冲突.假设有

子路径 $sp_1 = \{1,2,5,6,9,10,\cdots\}$,其中 1→2 与 9→10 是分支.如果分支 1→2 表明 $x > 3$,而分支 9→10 表明 $x < 2$,子路径 sp_1上,两个分支之间没有对相关变量 x 重新定值,则子路径 sp_1不可达.

(2)相关变量定值点与分支语句的特定分支条件有冲突.假设有子路径 $sp_2 = \{\cdots 9,12,13,14,15,\cdots\}$,其中 14→15 是分支,第 12 行是相关变量 y 的定值点.如果分支 14→15 表明 $y > 0$,而第 12 行将相关变量 y 的值定为 -5,且第 13 行没有对相关变量 y 重新定值,则子路径 sp_2不可达.

本阶段从子路径集中取出每一条子路径,分析其分支语句谓词与相关变量定值点,抽取路径条件,并将路径条件抽象为不等式组,通过约束求解器求解,根据求解结果,判断是否有以上两种情况出现,以判断子路径可达性.如果约束求解器的求解结果是可满足的,则该子路径为可达,否则该子路径为不可达.针对某一条子路径,如果求解结果为未知,则该子路径的可达性为无法判定.

计算控制依赖关系可以简化控制流图中控制流的分析过程.因此,SPE-IPD 根据程序的后支配树,结合程序的控制流图,计算出程序语句间的控制依赖关系.SPE-IPD 使用控制依赖树表示程序语句间的控制依赖关系.控制依赖树中后继节点代表的程序语句控制依赖于直接前驱节点与间接前驱节点代表的程序语句.得到控制依赖关系后,SPE-IPD 根据不等式组构造算法对每一条子路径构造相应的不等式组.

算法 2　不等式组构造算法

```
algorithm:inequality set generation
input:sp                              //待构造不等式组的子路径
      controlDependenceTree           //程序控制依赖树
output:inequalitySet
begin
1.  inequalitySet←Ø
2.  sp←sp.reverse
3.  foreach node in sp
4.    if node is in controlDependenceTree
5.      && nodehas more than 1 successors then
6.    inequalitySet←inequalitySet∪node.getBranchPredicate
7.    variableSet←variableSet∪node.getVariable
8.  else if node is a define statement
9.            && node.getVariable∈variableSet
10.           && node is first found
11.       inequalitySet←inequalitySet∪node.getDefineExpression
12.    endif
13.  endfor
end
```

不等式组构造算法逆向遍历子路径中所有的语句节点(第 2,3 行),并判断节点是否是分支语句节点或是相关变量定值节点.

如果节点 m 在控制依赖树中且有多于一个的后继节点(第 4,5 行),则节点 m 是一个分支语句节点.其后继节点代表的语句控制依赖于节点 m 代表的语句.SPE-IPD 提取出分支节点 m 的谓词 p,并把谓词 p 抽象为表示该分支节点的不等式,并加入到不等式集中(第 6 行).如果谓词 p 有相关变量 x,则将相关变量 x 放入 variableSet 中(第 7 行).

如果节点 n 是一个定值节点,且定值的变量是一个相关变量(节点 n 定义的变量在 variableSet 中),且定值关系是逆向遍历过程中首次遇到的(第 8 - 10 行),则称节点 n 是一个相关变量定值节点.SPE-IPD 将此定值节点中的定值表达式抽象为代表相关变量定值点的不等式,并加入不等式集中(第 11 行).

3.3　扩展子路径集

本阶段扩展子路径集,完善不可达路径的检测结果.SPE-IPD 在第 1 阶段生成的是子路径集,为得到程序中所有路径的可达性信息,在本阶段,根据子路径集扩展算法,将把子路径集中的每一条子路径扩展为路径,并判断路径的可达性,完善检测结果.

算法 3　子路径集扩展算法

```
algorithm:expand sub-path set
input:subPathSet              //子路径集
      CFGHeadSet              //控制流图入口节点集
output:infeasiblePathSet
       unknownPathSet
begin
1.  infeasiblePathSet←Ø
2.  unknownPathSet←Ø
3.  foreach sp in subPathSet
4.    head←sp.head
5.    foreach sp' in subPathSet
6.      if sp' contains head && sp'.head∈CFGHeadSet then
7.        fp←sp'.subList(0,head)∪sp
8.        if sp is infeasible || sp' is infeasible then
9.          infeasiblePathSet←infeasiblePathSet∪fp
10.       else
11.         unknownPathSet←unknownPathSet∪fp
12.       endif
13.     endif
14.   endfor
15. endfor
end
```

子路径集扩展步骤:首先取出一条子路径 sp,算法第 4 行取出子路径 sp 的首节点 head,然后在子路径集中遍历其他子路径并判断是否包含 head 节点(第 6 行),如果子路径 sp'包含 head 并且 sp'的首节点是控制流图的入口节点,则说明 sp'与 sp 可以连接为一条路径 fp(第 7

行).在 sp'中取出从首节点到 head 节点的部分序列,与 sp 连接为一条 fp,连接点为 sp'与 sp 的公共节点 head.最后根据子路径 sp 与 sp'的可达性将路径 fp 归入到相应的结果集中.如果 sp 或 sp'不可达,扩展出的路径 fp 也不可达;如果 sp 与 sp'均为可达或是无法判定,则无法直接判断 fp 的可达性,需要对其进行二次检测.

4 实验

在本文方法的基础上,实现了一个针对 Java 程序的不可达路径检测工具.该工具首先通过 Java 程序分析框架 Soot* 获得程序的控制流图,根据子路径集构造算法构造子路径集,完成第 1 阶段的工作;然后分析分支语句谓词与相关变量定值点并构造不等式组,并使用 SMTInterpol** 求解,判断子路径的可达性,完成第 2 阶段的工作;最后对子路径集进行扩展,获得所有路径的可达性信息,完成第 3 阶段的工作.最终将检测结果存储在 Sqlite*** 数据库中.

4.1 实验设计

为验证本文方法的准确性,设计第一组实验.我们对小规模基准程序进行检测.程序规模较小,便于手工分析源代码,找出其中的不可达路径,并与 SPE-IPD 检测结果进行对比,采用准确率与召回率评价本文方法的效果.

为验证本文方法的有效性,设计第二组实验.我们对规模较大的程序进行检测.程序规模较大,在实验中更容易得到数目可观的检测结果.为充分验证本文方法有效性,我们与 Java 不可达代码检测工具 Joogie[1] 进行了对比.

4.2 实验对象

第一组实验的实验对象为基准程序.实验对象如表 1 中 Group 1 所示.

表 1 实验对象

Group	Program	Description	Lines of code
1	Zip viewer	Zip 工具	65
	Bubble sort	冒泡排序	27
	Square root	求平方根	80
	Binary search	二分查找	31
	Triangle classifier	三角形分类	70
2	Ant	部署工具	80500
	Siena	网络事件通知	6035
	Jtopas	文本解析	5400
	Nanoxml	XML 解析	7646
	Xmlsecurity	XML 加密	16800

第二组实验的实验对象是 SIR**** 网站上的几组程序.实验对象如表 1 中 Group 2 所述.表中 Lines of code 是源代码的总行数,该数据来源于 SIR 网站.实验中选取 Ant 原始版本 v0,Siena 原始版本 v4,Jtopas 原始版本 v3,Nanoxml 原始版本 v5,Xmlsecurity 原始版本 v0 作为实验对象.

4.3 实验 1

表 2 为第一组实验结果.表中第 2 列是 SPE-IPD 检测出的不可达路径的数目;为验证 SPE-IPD 的准确性,我们手工分析了程序中真实存在的不可达路径数目,与 SPE-IPD 的检测结果做对比,如表中第 3 列所示;表中第 4 列是 SPE-IPD 误报的不可达路径的数目;第 5 列是检测的准确率,第 6 列是检测的召回率,其中准确率和召回率分别按式(1)和式(2)计算.

$$\mathrm{Pre}=\frac{I_{\mathrm{detected}}I_{\mathrm{false}}}{I_{\mathrm{detected}}}\times 100\% \tag{1}$$

$$\mathrm{Rec}=\frac{I_{\mathrm{detected}}I_{\mathrm{false}}}{I_{\mathrm{truely}}}\times 100\% \tag{2}$$

准确率体现了检测结果的准确性.召回率体现了检测结果的全面性.

表 2 第一组实验结果

Prog	I_{detected}	I_{true}	I_{false}	Pre	Rec
Zip viewer	2	5	0	100%	40%
Bubble sort	2	2	0	100%	100%
Square root	1	4	0	100%	25%
Binary search	4	4	0	100%	100%
Triangle classifier	41	41	0	100%	100%

从表 2 中实验结果可以看出 SPE-IPD 准确率很高,对于检测出的不可达路径,经验证确实存在于程序中.Bubble sort,Binary search,Triangle classifier 中的分支条件均为纯数值比较类型,约束求解器可以准确的求解,因此准确率与召回率均比较高.然而对于 Zip viewer,Square root,召回率较低,即 SPE-IPD 存在漏报的情况.经分析,Zip viewer 中包含对文件迭代器的判断;Square root 中包含对 Java 对象是否为空的判断,上述两种约束均无法使用约束求解器求解.因此产生了漏报.在不可达路径检测中,这一类约束条件是难以判断求解的[2].

4.4 实验 2

表 3 为第二组实验结果.表中 M_{checked}(Checked methods)是检测出的包含不可达路径的方法数目,$\mathrm{LOC}_{\mathrm{checked}}$(Checked lines of code)是检测出的方法中源代码总行数,表中 TP(Total paths)是检测出的方法中总路径数目.SPE-IPD 子表中的 I_{detected}(Infeasible paths detected)是 SPE-IPD 检测出的不可达路径数目,B_{detected}(Basic blocks detected)是 Joogie 工具检测出的不可达基本块的

* http://www.sable.mcgill.ca/soot/

** http://ultimate.informatik.uni-freiburg.de/smtinterpol/

*** http://www.sqlite.org/

**** http://sir.unl.edu/portal/index.php

数目,Joogie 子表中的 $I_{detected}$(Infeasible paths detected)是我们统计出的,经过不可达基本块的不可达路径数目.

由表 3 实验结果可知,对于规模较大的程序,SPE-IPD 可以检测出数目非常可观的不可达路径.Ant 功能复杂,实验中检测出的方法大多分布在 org.apache.tools.ant.taskdef 包;Jtopas 与 Nanoxml 分别涉及到文本文件与 XML 文件的解析;XMLsecurity 涉及到 XML 文件的解析与加密,以上四个程序存在大量的字符串或纯数值比较分支,属于约束求解器可以求解的数据类型,因此可以检测出数目可观的不可达路径.Siena 用于网络事件通知,虽然也包含大量的分支循环语句,但限于约束求解器的局限性,可以求解的约束所占比例并不高,因此检测结果较少.

对于 Joogie 工具,检测出的是程序中不可达基本块.相比 SPE-IPD,Joogie 工具检测的粒度更小.对于实验对象 Siena 与 Nanoxml,Joogie 检测失败.原因是 Joogie 工具在检测之前,首先把 Java 代码转换为 Boogie 代码,尚有一些 Java 语句与数据类型无法转换到 Boogie 代码,因此检测失败.

表 3　第二组实验结果

Prog	SPE-IPD					Joogie					
	$M_{checked}$	$LOC_{checked}$	TP	$I_{detected}$	Time	$M_{checked}$	$LOC_{checked}$	TP	$B_{detected}$	$I_{detected}$	Time
Ant	85	1958	31273	20427	17300s	33	940	9437	68	12366	4116s
Siena	3	33	46	21	1287s	–	–	–	–	–	–
Jtopas	40	690	1054	749	1123s	4	78	63	6	196	297s
Nanoxml	21	541	8987	5393	1570s	–	–	–	–	–	–
Xmlsecurity	44	868	26188	8914	3310s	12	291	17403	19	5610	834s

Joogie 检测出的是程序中的不可达基本块,SPE-IPD 检测出的是程序中的不可达路径.我们统计出经过不可达基本块 $B_{detected}$的不可达路径 $I_{detected}$,与 SPE-IPD 检测出的不可达路径数目进行了比较,同时也比较了检测出的包含不可达路径的方法数量.可以发现,本文方法 SPE-IPD 检测效果好于 Joogie,而且对于 Joogie 无法检测的 Siena 与 Nanoxml,SPE-IPD 均可成功完成检测工作,因此,本文方法 SPE-IPD 是非常有效的.

我们从 35504 条不可达路径检测结果中抽取 500 条不可达路径,分析无法求解的冲突原因,并与程序中实际情况进行校验与对比,发现检测出的冲突原因与真实情况相符.

表 3 中包含了实验消耗的时间(Time).实验消耗的时间包含以下几个部分.首先是 Soot 分析 Java 类的时间,Soot 分析一个类大约耗时 8s.其次是生成子路径集的时间,我们采用递归的方式生成子路径集,每千行代码大约耗时 500ms.约束求解器求解包含 5 个方程的二元一次方程组大约耗时 250ms,可以作为判断子路径可达性所需的平均时间.子路径集扩展算法复杂度较高,与待测程序的结构与规模有关,难以估计出平均消耗时间,我们采用了多线程技术加快子路径集扩展速度.为了方便对检测结果进行查看与检索,我们将检测结果写入到 Sqlite 数据库中,由于 Sqlite 数据库写入速度的限制,此阶段是非常耗时的阶段,约消耗实验中接近一半的时间.综上所述,在保证检测结果全面性的前提下,本文方法所消耗的时间代价是比较合理的.

表 4 展示了检测出包含不可达路径的方法中子路径集的信息.TSP(Total sub-paths)是方法中包含的子路径数目.$FSP_{detected}$(Feasible sub-paths detected)是子路径集中可达的子路径数目.$ISP_{detected}$(Infeasible sub-paths detected)是子路径集中不可达的子路径数目.$UNSP_{detected}$(Unknown sub-paths detected)是子路径集中可达性未知的子路径数目.DCR(Double check rate)是二次检查率,即检测出的不可达路径中,需要二次检查确定其不可达的路径所占的比例.由表 4 可知,方法中子路径的数量一般占所有路径数量的 10%左右.对于检测出的包含不可达路径的方法,子路径集中不可达子路径所占的比例约为 50%,对于这类子路径扩展出的路径,不需要做二次检测工作.由表 4 中 DCR 数据可知,实验中约有 70%的不可达路径是在对子路径集的检测过程中获得的,即仅有 30%的不可达路径是在二次检测中检测出的.综上所述,本文方法可以准确有效地检测出程序中的不可达路径,减少检测路径的数量,提高不可达路径检测效率,节约测试资源.

表 4　子路径集信息

Prog	TSP	$FSP_{detected}$	$ISP_{detected}$	$UNSP_{detected}$	DCR
Ant	4690	374	3064	1252	37.0%
Siena	6	2	1	3	19.0%
Jtopas	110	7	67	36	23.6%
Nanoxml	855	66	552	237	25.9%
Xmlsecurity	2530	334	903	1293	34.8%

5　相关工作

文献[3~5]提出基于分支相关性的检测方法,这类方法对于复杂的谓词检测精度不高,存在一定的局限性,本文方法使用约束求解器检测谓词,对于一阶谓

词,检测结果较为精确.

文献[6~8]提出基于路径条件满足性的检测方法,使用近似于符号执行的方式抽取路径条件,并通过判断路径条件检测路径的可达性.然而,这类方法以路径作为分析对象,分析代价较高,本文方法主要以子路径作为分析对象,降低了分析代价.

文献[9~11]提出动态检测方法,在限定搜索深度内针对路径做测试用例生成工作,如果无法找到覆盖该路径的测试用例,则将路径标记为不可达.这种方法在以覆盖基路径为目标的测试中十分有效,但如果针对程序中所有路径做检测工作,仍将面临路径爆炸问题,检测代价较高,本文方法检测范围更为全面,同时检测工作主要在子路径集上完成,有效地解决了路径爆炸问题的同时,又可以获得程序中所有路径的可达性信息.

文献[12]提出一种动静结合的方法,该方法结合了静态的分支相关性分析方法与动态的主动选取可达路径的方法,达到了不错的检测效果,然而由于该检测方法结合了动态分析与静态分析,检测开销较大,不适合检测规模较大的程序,本文方法是静态分析方法,不需要执行被测程序,避免了动态分析所需的开销.

6 结束语

提出一种基于子路径扩展的不可达路径静态检测方法 SPE-IPD.该方法首先提取出程序的子路径集,分析分支语句谓词与相关变量定值点,构造不等式组,然后通过约束求解器进行求解,根据求解结果,判断子路径是否可达,将不可达的子路径扩展为不可达路径,对可达或是无法判定的子路径扩展出的路径做二次检测工作以判断其可达性,最终得到程序中所有路径的可达性信息.实验结果表明,本文方法准确性较高,可以有效地检测出程序中的不可达路径.

参考文献

[1] Arlt S, Sch F M. Joogie: Infeasible code detection for Java[A]. Proceedings of CAV'12[C]. Berlin: Springer, 2012. 767 – 773.

[2] Ding Sun, Zhang Hongyu, Tan H B K. Detection of infeasible paths: Approaches and challenges[A]. Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering[C]. Berlin: Springer, 2013. 64 – 78.

[3] Gong Dun Wei, Yao Xiang Juan. Automatic detection of infeasible paths in software testing[J]. IET Software, 2010, 4(5): 361 – 370.

[4] Zhuang Xiao Tong, Zhang Tao, Pande S. Using branch correlation to identify infeasible paths for anomaly detection[A]. Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture[C]. Washington, DC: IEEE Computer Society, 2006. 113 – 122.

[5] Suhendra V, Mitra T, Roychoudhury A, et al. Efficient detection and exploitation of infeasible paths for software timing analysis[A]. Proceedings of the 43rd Annual Design Automation Conference[C]. New York: ACM, 2006. 358 – 363.

[6] Delahaye M, Botella B, Gotlieb A. Explanation-based generalization of infeasible path[A]. Proceedings of the 3th International Conference on Software Testing, Verification and Validation[C]. Los Alamitos: IEEE, 2010. 215 – 224.

[7] Jaffar J, Murali V, Navas J, et al. TRACER: A symbolic execution tool for verification[A]. Proceedings of CAV'12[C]. Berlin: Springer, 2012. 758 – 766.

[8] Tomb A, Flanagan C. Detecting inconsistencies via universal reachability Analysis[A]. Proceedings of ISSTA'12[C]. New York: ACM, 2012. 287 – 297.

[9] Hermadi I, Lokan C, Sarker R. Dynamic stopping criteria for search-based test data generation for path testing[J]. Information and Software Technology, 2014, 56(4): 395 – 407.

[10] Ghiduk A S. Automatic generation of basis test paths using variable length genetic algorithm[J]. Information Processing Letters, 2014, 114(6): 304 – 316.

[11] Tonella P, Tiella R, Nguyen C D. Interpolated N-grams for model based testing[A]. Proceedings of ICSE'14[C]. New York: ACM, 2014. 10 – 21.

[12] Ngo M N, Tan H B K. Heuristics-based infeasible path detection for dynamic test data generation[J]. Information and Software Technology, 2008, 50(7): 641 – 655.

作者简介

王红阳 男,1990 年出生,黑龙江伊春人.中国矿业大学研究生.研究方向为软件测试、不可达路径检测.

E-mail: wanghongyang@cumt.edu.cn

姜淑娟(通信作者) 女,1966 年出生,山东莱阳人.中国矿业大学教授,博士生导师,CCF 会员.主要研究领域为编译技术、软件工程等.

E-mail: shjjiang@cumt.edu.cn