软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(5):1450−1463 [doi: 10.13328/j.cnki.jos.005720]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

敏感变量和感知机结合的测试预言生成方法*

马春燕, 李尚儒, 王慧朝, 张 磊, 张 涛

(西北工业大学 软件与微电子学院,陕西 西安 710072)

通讯作者: 马春燕, E-mail: machunyan@nwpu.edu.cn

摘 要: 测试预言生成技术是软件工程测试领域的研究热点之一.没有可以利用的历史测试用例是目前大部分测试预言生成技术的普遍假设,但是这种假设会使我们错过利用现有部分测试用例协助自动生成新测试用例预言的机会.在已知部分测试用例集的情况下,提出了基于敏感变量和线性感知机的新测试用例的测试预言自动生成方法.首先,收集已知测试用例执行的语句覆盖和不同断点处内存值集合作为训练集,计算与新测试用例执行覆盖信息高度相似的测试用例集;其次,计算各断点处表征成功或失败的敏感变量集;然后,应用线性感知机求解每个断点处成功或失败概率预测的门限值;在此基础上,给出新测试用例测试预言自动生成的方法,并对方法进行讨论;最后,采用7个程序的129个故障版本作为实验对象,对共计14 300个测试用例生成测试预言.实证评价表明,测试预言准确率平均达到96.2%.该成果可以形成测试用例集合构造的"滚雪球效应",不断迭代自动生成新测试用例的测试预言.

关键词: 测试预言;线性感知机;敏感变量;测试用例;内存值

中图法分类号: TP311

中文引用格式: 马春燕,李尚儒,王慧朝,张磊,张涛.敏感变量和感知机结合的测试预言生成方法.软件学报,2019,30(5):1450−1463. http://www.jos.org.cn/1000-9825/5720.htm

英文引用格式: Ma CY, Li SR, Wang HC, Zhang L, Zhang T. Generation method for test oracle based on sensitive variables and linear perceptron. Ruan Jian Xue Bao/Journal of Software, 2019,30(5):1450−1463 (in Chinese). http://www.jos.org.cn/1000-9825/5720.htm

Generation Method for Test Oracle Based on Sensitive Variables and Linear Perceptron

MA Chun-Yan, LI Shang-Ru, WANG Hui-Chao, ZHANG Lei, ZHANG Tao

(School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: Test oracle generation technology is one of the hotspots in the testing field of software engineering. There are no historical test case sets available, which are common assumptions about existing test oracle generation techniques. However, this assumption may not always hold, and where it does not, there may be a missed opportunity; perhaps the pre-existing test cases could be used to assist the automated oracle generation of new test cases. In the case of the existing test case set, an automatic test oracle generation method for a new test case based on sensitive variables and linear perceptrons is proposed. Firstly, the statement coverage and memory value set executed by some known test cases are collected, and a set of test cases with high similarity to the execution coverage information of the new test case is computed. Secondly, the memory sensitive variable set solving algorithm of the program at a given breakpoint is given. Thirdly, the known test case set as the training set and the perceptron is used to solve the threshold value at each breakpoint. And on this base an automatic oracle generation method of the new test case is proposed. Finally, 129 fault versions of seven programs are used as experimental objects to generate test oracles of 14 300 new test cases. The empirical evaluation shows that the generated test oracle can

* 基金项目: 国家自然科学基金(61103003); 航天基金(2018KC160026); 上海航天科技支撑计划(2017MC160014)

Foundation item: National Natural Science Foundation of China (61103003); Aerospace Fund (2018KC160026); Shanghai Aerospace Science and Technology Support Program (2017MC160014)

本文由智能化软件新技术专刊特约编辑申富饶教授和李戈副教授推荐.

收稿时间: 2018-08-31; 修改时间: 2018-10-31; 采用时间: 2018-12-13

achieve 96.2% accuracy on average. The results of the research can form the “snowball effect” of the test case set construction, and iteratively automatically generate test oracles for new test cases.

Key words: test oracle; linear perceptron; sensitive variables; test cases; memory value

软件测试是保证软件系统质量的主要活动.在软件测试中,测试预言的作用至关重要,它是一种判断程序在给定测试输入下的执行结果是否符合预期的方法.测试预言的质量直接影响测试活动的有效性和软件系统的质量.目前,虽然研究人员已提出了各种自动化测试输入生成技术,成果颇丰,但是测试预言问题仍然被公认为是软件测试中最难解决的问题之一[1].

测试预言问题的综述文献[1]将测试预言生成的研究分为基于软件形式化规格说明或形式化模型(例如状态机或时序逻辑规范等)的测试预言生成方法(317 篇相关论文)、基于各种软件制品(例如需求和设计文档、软件执行的属性信息、软件其他版本等)的测试预言生成方法(245 篇相关论文)、基于常识或隐含的知识生成测试预言(76 篇相关论文)这 3 类.第 1 类需要软件开发人员或测试人员给出软件的形式化规格说明,人工参与的工作量较大,形式化规格说明缺乏以及程序行为描述不全面等,是该类方法面临的极大挑战;第 2 类研究克服了第 1 类研究面临的挑战,正在成为一个活跃的研究方向,主要研究包括基于规格说明或编程接口挖掘的方法、基于程序文档的挖掘方法、蜕变测试、基于源码的静态和动态分析方法以及基于机器学习的方法等,该类测试预言的生成方法包括全自动的方法或人工辅助的半自动化方法,其中,半自动化方法较多;第 3 类应用领域较窄,仅针对特种类型或特定的应用范围的软件.

智能化技术在测试预言生成方面的应用研究具有鲜明的学科交叉特点,目前研究成果相对较少,上述第 2 类测试预言生成方法的研究中逐步开始出现智能化技术的应用.本文的工作属于无需测试人员干预的测试预言自动生成方法,借助揭示成功和失败概率的敏感变量,采用监督的机器学习算法——线性感知机作为测试预言自动生成的模型,收集部分测试用例的语句覆盖、不同断点处的内存变量及其取值的集合等数据信息作为训练集,观察新测试用例执行的相应语句覆盖和内存变量集等信息,对该新测试用例在不同断点处的执行结果是成功还是失败进行预测,自动生成测试预言.本文成果可以形成测试用例集合构造的“滚雪球效应”,不断迭代自动生成新测试用例或回归测试用例的测试预言.

本文第 1 节首先给出相关定义和假设,在此基础上,给出求解断点处敏感变量集的算法,并应用线性感知机算法求解断点处成功或故障概率的门限值,提出本文的测试预言生成算法.第 2 节对算法进行讨论.第 3 节给出实验对象和实验方法.第 4 节通过案例对算法进行阐释和验证.第 5 节将已有研究工作进行总结,并与本文工作进行对比分析.

1 测试预言自动生成方法

1.1 相关定义

本文测试预言生成方法用到的定义如下.

定义 1(测试用例 t_a 和 t_b 的相似度 $Sim(t_a,t_b)$). 设测试用例 t_a 和 t_b 执行的语句集合分别为 A(其元素个数记作$|A|$)和 B(其元素个数记作$|B|$),$|A\cap B|$表示 A 与 B 交集的元素个数,则 t_a 和 t_b 的相似度定义为 $Sim(t_a,t_b)=2\times|A\cap B|/(|A|+|B|)\in[0,1]$.$Sim(t_a,t_b)$的值越大,$t_a$ 和 t_b 的相似度越高:如果 $A=B$,则 $Sim(t_a,t_b)=1$,t_a 和 t_b 的相似度最高;如果 $A\cap B=\varnothing$,则 $Sim(t_a,t_b)=0$,t_a 和 t_b 的相似度最低.

定义 2(内存变量). 内存变量是一个二元式$[var,val]$,其中,var 表示变量名,val 表示该变量的值.

定义 3(两个内存变量相等). 两个内存变量$[var_a,val_a]$和$[var_b,val_b]$相等当且仅当 $var_a=var_b$ 和 $val_a=val_b$.

定义 4(两个内存变量集合 M 和 N 的交集). 两个内存变量集合 M 和 N 的交集记作 $M\cap N$,定义为 M 与 N 中相等的内存变量的集合.

定义 5(两个内存变量集合 M 和 N 的差集). 两个内存变量集合 M 和 N 的差集记作 $M-N$,定义为与 N 中变量名相同而变量值不同的 M 中内存变量的集合.

定义 6(敏感变量的定义). 一个敏感变量 t 是一个内存变量二元式 $t=[sv,v]$,其中,sv 表示敏感变量的名字,v 表示敏感变量的值.

1.2 相关假设:

根据程序测试和调试经验,本文的算法思想基于下述基本假设.

- 假设 1:变量集对成功和失败具有一定的揭示能力.

对于高相似度的测试用例(见定义 1)形成的一个集合 St 而言,在每个断点处,St 中测试用例执行的变量集合的交集或差集对程序执行的成功概率或失败概率具有一定的揭示能力.

- 假设 2:不同断点处变量集合对成功或故障的揭示能力不同.

因为故障发生程序的某个位置且故障有传播能力,断点位置越靠后,其相应变量集合对成功或故障的揭示能力逐步增强.

1.3 测试预言生成算法

已知一个程序、一个测试用例集合及各测试用例执行成功或失败的信息,本文通过在程序中设置批量断点的方法收集所有测试用例在各断点处的内存数据以及代码覆盖信息作为训练集;然后根据新测试用例的执行,计算各断点处的敏感变量集合,并应用线性感知机算法计算该断点处执行失败或成功概率的门限值;在此基础上,本节给出了一种基于内存分析的测试预言自动生成方法,即对新产生的测试用例执行同一程序的运行结果是成功还是失败进行预测,自动生成测试预言(见下文算法 1).为了阐述本文提出的测试预言生成算法,表 1 给出了算法中涉及的符号名称及其含义描述.

Table 1 Notations and their meanings in this section's algorithm

表 1 本节算法中涉及的符号名称及其含义描述

符号名称	英文全称	含义描述
P	Program	故障程序
ST	Set of n test cass	n 个测试用例的集合$\{[t_1,o_1],[t_2,o_2],[t_3,o_3],\ldots,[t_n,o_n]\}$ t_i 是第 i 个测试输入,o_i 是第 i 个测试输入的执行结果(成功或失败),o_i 为 0,表示 t_i 执行失败;o_i 为 1,表示 t_i 执行成功.在算法描述中,也用 T_i 表示第 i 个测试用例
SB	Set of m breakpoints	程序 P 中设置的 m 个断点集合$\{b_1,b_2,b_3,\ldots,b_m\}$,$b_i$ 表示第 i 个断点
SSB	Set of all sensitive variables for m breakpoints	程序 P 中 m 个断点处各敏感变量集合构成的集合$\{SB_{b1},SB_{b2},SB_{b3},\ldots,SB_{bm}\}$,$SB_{bi}$ 表示第 i 个断点 b_i 处的敏感变量集合
t_{new}	–	表示一个待生成测试预言的新测试用例$[t,o]$,t 是测试输入, o 为 0,表示 t 执行失败;o 为 1,表示 t 执行成功
PP_i	Propobility of passing for t_{new} under breakpoint b_i	P_i 表示 t_{new} 第在 $i(1\leqslant i\leqslant m)$个断点 b_i 处执行成功的概率
PP	Propobility of passing for t_{new}	表示 t_{new} 执行成功的概率
STt_{new}	Set of test cases that are most similar to t_{new}	表示 ST 中与 t_{new} 最相似的测试用例集
$SPTt_{new}$	Set of passing test cases in STt_{new}	表示 STt_{new} 中成功测试用例组成的集合
$SFTt_{new}$	Set of failing test cases in STt_{new}	表示 STt_{new} 中失败测试用例组成的集合

1.3.1 算法 1:测试预言生成算法

给定 P,ST,SB 和 t_{new} 的测试输入 t,首先计算 ST 中与 t_{new} 最相似的测试用例集合 STt_{new}(见下述算法 1 伪代码中的第 1 行~第 20 行).将获得的 STt_{new} 分为最相似成功测试用例集 $SPTt_{new}$ 和失败测试用例集 $SFTt_{new}$,调用后文算法 2 计算 SB 中每个断点处的敏感变量集.

- 如果 $SPTt_{new}\neq\varnothing$,$SFTt_{new}\neq\varnothing$,求得的敏感变量的含义为:从所有与 t_{new} 相似正确测试用例和失败测试用例的执行来看,可能错误的内存变量有哪些,在这种情况下,敏感变量称为“故障敏感变量”;
- 如果 $SPTt_{new}\neq\varnothing$,$SFTt_{new}=\varnothing$,求得的敏感变量的含义为:从所有与 t_{new} 相似的正确测试用例的执行来看,可能正确的内存变量有哪些,在这种情况下,敏感变量称为“成功敏感变量”;
- 如果 $SPTt_{new}=\varnothing$,$SFTt_{new}\neq\varnothing$,求得的敏感变量的含义为:从所有与 t_{new} 相似的失败测试用例的执行来看,

可能错误的内存变量有哪些,在这种情况下,敏感变量称为“故障敏感变量”.

然后调用后文算法 3,计算 b_i 处的门限值 $Threshold_{bi}$,根据算法 1 的第 21 行~第 79 行陆续计算各断点处程序执行成功的概率 $PP_1,PP_2,PP_3,\ldots,PP_m$.

最后,对于序列 $PP_1,PP_2,PP_3,\ldots,PP_m$,用等差序列作为对预测结果影响所占的权重,分别记为 $a_1,a_2,a_3,\ldots,a_m$,则等差数列的公差为 $d=(a_m-a_1)/(m-1)$,可以求得该等差数列首项 $a_1=1/m-(a_m-a_1)/2=3/4m$,其余各项为 $a_i=a_1+(i-1)d$,测试用例执行程序 P 成功的概率为 $PP=PP_1*a_1+PP_2*a_2+\ldots+PP_m*a_m$.

算法 1 的伪代码如下所示(第 1 行~第 86 行).

算法 1. *GeneratingTestOracles*.

Input: *P*, *ST*, *SB* 和 the test input of t_{new}.

Output: Test oracle of t_{new}.

1. $temp\leftarrow t_{new}$
2. $STt_{new}\leftarrow\varnothing$
3. **While** (STt_{new}!=∅)
4. $C\leftarrow\varnothing$
5. $MaxSimularTestCases\leftarrow\varnothing$
6. $Largenumber\leftarrow 0$
7. **for** (each t_i in *ST*)
8. **if** $Sim(temp,t_i)=1$
9. $C\leftarrow C\cup\{t_i\}$
10. $STt_{new}\leftarrow STt_{new}\cup C$
11. **end for**
12. **if** ($C=\varnothing$)
13. **for** (each t_i in *ST*)
14. **if** ($Sim(temp,t_i)>largenum$)
15. $largenum\leftarrow Sim(temp,t_i)$
16. $temp\leftarrow t_i$
17. **End if**
18. **end for**
19. **end if**
20. **end wile**
21. $SPTt_{new}\leftarrow$Select the set of the passing test cases from STt_{new}
22. $SFTt_{new}\leftarrow$Select the set of the failing test cases from STt_{new}
23. **for** (each b_i in *SB*)
24. $SB_{bi}\leftarrow CalculateSensitiveVariabls(P,SPTt_{new},SFTt_{new},b_i)$
25. **Endfor**
26. $m\leftarrow 0$
27. **for** (each b_i in *SB*)
28. $Threshold_{bi}\leftarrow Threshold(P,SPTt_{new},SFTt_{new},b_i,SB_{bi})$
29. $FC_i\leftarrow 0$
30. $PC_i\leftarrow 0$
31. **if** ($SPTt_{new}$!=∅,$SFTt_{new}$!=∅)
32. **for** (each test case t_j in $SPTt_{new}$)

33. M_t←get memory variables set from the execution of t_{new} for b_i
34. M_{tj}←get memory variables set from the execution of t_j for b_i
35. XC_j←$(M_t–M_{tj})\cap SB_{bi}$
36. **If** ($|XC_j|/|(M_t–M_{tj})|>Threshold_{bi}$)
37. $FC_i=FC_i+1$
38. **else**
39. $PC_i=PC_i+1$
40. **end if**
41. **end for**
42. **for** (each test case t_j in $SFTt_{new}$)
43. M_t←get memory variables set from the execution of t_{new} for b_i
44. M_{tj}←get memory variables set from the execution of t_j for b_i
45. XC_j←$M_t\cap M_{tj}\cap SB_{bi}$
46. **If** ($|XC_j|/|(M_t–M_{tj})|>Threshold_{bi}$)
47. $FC_i=FC_i+1$
48. **Else**
49. $PC_i=PC_i+1$
50. **end if**
51. **end for**
52. **else if** ($SPTt_{new}$!=∅,$SFTt_{new}$=∅)
53. **for** (each test case t_j in $SPTt_{new}$)
54. M_t←get memory variables set from the execution of t_{new} for b_i
55. M_{tj}←get memory variables set from the execution of t_j for b_i
56. XC_j←$M_t\cap M_{tj}\cap SB_{bi}$
57. **if** ($|XC_j|/|(M_t–M_{tj})|<Threshold_{bi}$)
58. $FC_i=FC_i+1$
59. **else**
60. $PC_i=PC_i+1$
61. **end if**
62. **end for**
63. **else if** ($SPTt_{new}$=∅,$SFTt_{new}$!=∅)
64. **for** (each test case t_j in $SFTt_{new}$)
65. M_t←get memory variables set from the execution of t_{new} for b_i
66. M_{tj}←get memory variables set from the execution of t_j for b_i
67. XC_j←$M_t\cap M_{tj}\cap SB_{bi}$
68. **If** ($|XC_j|/|(M_t–M_{tj})|>Threshold_{bi}$)
69. $FC_i=FC_i+1$
70. **else**
71. $PC_i=PC_i+1$
72. **end if**
73. **end for**
74. **end if**

75. $PP_i \leftarrow PC_i/(PC_i+FC_i)$
76. $m \leftarrow m+1$
77. **End for**
78. $d \leftarrow (a_m-a_1)/(m-1)$
79. $a_1 \leftarrow 1/m-(a_m-a_1)/2=3/4m$
80. $a_i \leftarrow a_1+(i-1)d$
81. $PP \leftarrow PP_1*a_1+PP_2*a_2+\ldots+PP_m*a_m$
82. **if** $PP>0.5$
83. **return** 1
84. **else**
85. **return** 0
86. **end if**

1.3.2 算法 2:敏感变量集求解算法

给定输入 P,$SPTt_{new}$,$SFTt_{new}$ 以及 SB 中第 i 个断点 b_i,设两个临时变量 ST_1 和 ST_2.

- 如果 $SPTt_{new}\neq\varnothing$,$SFTt_{new}\neq\varnothing$,则令 $ST_1=SPTt_{new}$,$ST_2=SFTt_{new}$;
- 如果 $SPTt_{new}\neq\varnothing$,$SFTt_{new}=\varnothing$,则令 $ST_1=SPTt_{new}$,$ST_2=SPTt_{new}$;
- 如果 $SPTt_{new}=\varnothing$,$SFTt_{new}\neq\varnothing$,则令 $ST_1=SFTt_{new}$,$ST_2=SFTt_{new}$.

令 $x=|ST_1|$,$y=|ST_2|$,首先计算得到个内存变量集合$\{d_{j1},d_{j2},d_{j3},\ldots,d_{jy}\}$,将其存入哈希表 D_{pj},其中,$d_{jk}(1\leqslant k\leqslant y)$作为哈希表的 *key*(记作 $d_{jk}.key$),该内存变量出现的次数作为哈希表的 *value*(记作 $d_{jk}.value$)

然后,可计算得到的 x 个哈希表 $D_{p1},D_{p2},D_{p3},\ldots,D_{px}$,计算哈希表 $H_{bi}=D_{p1}\cup D_{p2}\cup\ldots\cup D_{pX}$,其中,$\cup$表示的含义为:各个 $D_{pj}(1\leqslant j\leqslant|SFTt_{new}|)$中,*key* 不同则合并;*key* 相同,则 *value* 相加.

最后对 H_{bi} 中各 *value* 值从大到小排序,*key* 值为前 20%的变量(取 *key* 值较大的变量作为敏感变量)加入 SB_{bi},算法结束,返回 SB_{bi}.

算法 2 的伪代码如下所示(第 1 行~第 29 行).

算法 2. *CalculateSensitiveVariabls*.

Input: P, $SPTt_{new}$, $SFTt_{new}$, and the i-th breakpoint b_i in SB.

Output: SB_{bi} which is the set of sensitive variables for the i-th breakpoint b_i in SB.

1. **if** ($SPTt_{new}$!=∅ && $SFTt_{new}$!=∅)
2. $ST_1 \leftarrow SPTt_{new}$
3. $ST_2 \leftarrow SFTt_{new}$
4. **else if** ($SPTt_{new}$!=∅,$SFTt_{new}$=∅)
5. $ST_1 \leftarrow SPTt_{new}$
6. $ST_2 \leftarrow SPTt_{new}$
7. **else if** ($SPTt_{new}$=∅,$SFTt_{new}$!=∅)
8. $ST_1 \leftarrow SFTt_{new}$
9. $ST_2 \leftarrow SFTt_{new}$
10. **end if**
11. $j \leftarrow 1$
12. **for** (each test case $t1_j$, from ST_1)
13. $k \leftarrow 1$
14. **for** (each test case $t2_k$, from ST_2)
15. $M \leftarrow$get memory variables set from the execution of $t1_j$

16. N←get memory variables set from the execution of $t2_k$
17. **if** (ST_1!=ST_2)
18. d_{jk}←M–N
19. **else if**
20. d_{jk}←M∩N
21. **end if**
22. *hashMap* D_{pj}(*key*,*value*)←(d_{jk}, the cumulative number of times of d_{jk} appearing)
23. k++
24. **end for**
25. j++
26. **end for**
27. H_{bi}←D_{p1}∪D_{p2}∪…∪D_{pX}, different keys are merged and the *value* is summed for the same *key*
28. SB_{bi}←selected the top 20% from the sorted H_{bi} according to the *value* in the hasmap
29. **return** SB_{bi}.

1.3.3 算法 3:应用线性感知机算法求解每个断点处门限值

给定输入 P,$SPTt_{new}$,$SFTt_{new}$,b_i 以及 SB_{bi},首先初始化候选门限值二维数组 *temp*[][]为 0,两维数组长度都为 $SPTt_{new}$∪$SFTt_{new}$ 中测试用例个数.对于 $SPTt_{new}$∪$SFTt_{new}$ 中每个测试用例 t_k(1≤k≤|$SPTt_{new}$∪$SFTt_{new}$|),将其作为一个新测试用例,分别计算 *temp* 的第 k 行元素,根据“测试预言生成算法”,对于 $SPTt_{new}$∪$SFTt_{new}$ 中每个测试用例 t_k(1≤k≤|$SPTt_{new}$∪$SFTt_{new}$|),将其作为一个新测试用例 t_{new};计算每个候选门限值的预测成功率,将 *temp* 中预测准确率最大的候选门限值存入 $Threshold_{bi}$,算法结束,返回 $Threshold_{bi}$.

算法 3 的伪代码如下所示(第 1 行~第 34 行).

算法 3. *Threshold*.

Input: P, $SPTt_{new}$, $SFTt_{new}$, b_i, SB_{bi}.

Output: $Threshold_{bi}$ for b_i.

1. max←|$SPTt_{new}$∪$SFTt_{new}$|
2. *temp*[max][max]←0
3. **for** (each test case t_k in STt_{new})
4. **if** ($SPTt_{new}$!=∅,$SFTt_{new}$!=∅)
5. **for** (each test case t_j in $SPTt_{new}$)
6. M_t←get memory variables set from the execution of t_{new} for b_i
7. M_{tj}←get memory variables set from the execution of t_j for b_i
8. XC_j←(M_t–M_{tj})∩SB_{bi}
9. *temp*[k][j]←|XC_j|/|(M_t–M_{tj})|
10. **end for**
11. **for** (each test case t_j in $SFTt_{new}$)
12. M_t←get memory variables set from the execution of t_{new} for b_i
13. M_{tj}←get memory variables set from the execution of t_j for b_i
14. XC_j←M_t∩M_{tj}∩SB_{bi}
15. *temp*[k][j]←|XC_j|/|(M_t∩M_{tj})|
16. **end for**
17. **else if** ($SPTt_{new}$!=∅,$SFTt_{new}$=∅)
18. **for** (each test case t_j in $SPTt_{new}$)

19. M_t←get memory variables set from the execution of t_{new} for b_i
20. M_{tj}←get memory variables set from the execution of t_j for b_i
21. XC_j←M_t∩M_{tj}∩SB_{bi}
22. $temp[k][j]$←$|XC_j|/|(M_t∩M_{tj})|$
23. **End for**
24. **else if** ($SPTt_{new}$=∅,$SFTt_{new}$!=∅)
25. **for** (each test case t_j in $SFTt_{new}$)
26. M_t←get memory variables set from the execution of t_{new} for b_i
27. M_{tj}←get memory variables set from the execution of t_j for b_i
28. XC_j←M_t∩M_{tj}∩SB_{bi}
29. $temp[k][j]$←$|XC_j|/|(M_t∩M_{tj})|$
30. **end for**
31. **end if**
32. **end for**
33. $Threshold_{bi}$←select the *value* that has the highest prediction accuracy in training set.
34. **return** $Threshold_{bi}$

2 实验对象及实验方法

本文采用 SIR(software-artifact infrastructure repository)知识库[2]提供的 5 个 C 程序实验对象和从开源网站 travis-ci 上下载的两个 C 程序实验对象 expresstionParser 和 sort(网址:https://travis-ci.org/swenson/),通过实际案例程序阐释本文给出的测试预言自动生成方法的有效性.表 2 给出了实例验证过程中所使用的每个案例名、功能、故障版本数、测试用例数以及正确版本的源码行数等信息.

Table 2 Experimental objects

表 2 实验对象

程序名	功能描述	故障版本数	测试用例数	代码行数
tcas	空中防撞系统	41	1 500	174
print_tokens	词法分析器	7	4 130	726
schedule2	优先级调度器	10	2 710	374
replace	模式替换	32	5 542	564
tot_info	信息统计	23	1 052	565
expressionParser	表达式解析和运算	8	1 500	1 251
sort	各类排序算法比较	8	1 500	2 512

为了将提出的测试预言生成算法进行实践,本文开发了图 1 所示的实验辅助工具,工具的开发环境是 Ubuntu Linux 10.04.为了方便与 SQLite 数据库以及 GDB 调试集成,本文应用 Python 语言和 shell script 实现工具的开发,辅助工具源码 TestOracleGen.rar 在 https://github.com/machunyan/LCEC 网站上可以公开下载.工具的主要模块功能阐述如下.

1) 程序代码分析模块.该模块负责分析源文件,根据断点的设置原则进行断点标注(本文实验过程中设置的断点设置原则为:以 return 和 exit 作为结束的行,本实验辅助工具允许用户批量自定义断点位置),为程序执行以及执行过程中调用 GDB 命令获取内存数据做准备.
2) 程序执行分析模块.该模块运行每一个测试用例,在程序运行过程中获取断点处程序执行的内存变量集,并通过 gcov 软件获取每个测试用例运行时的程序覆盖信息.
3) 数据存储模块.该模块将程序代码分析模块和程序执行分析模块得到的断点信息和测试用例的执行覆盖信息存入数据库.在分析所有测试用例执行覆盖的基础上,对不同覆盖进行分类,计算测试用例

之间执行覆盖的联系.

4) 预言模块.该模块根据上述 3 个模块收集的故障程序的断点信息、测试用例运行故障程序时所有断点的内存变量信息以及执行语句覆盖信息,执行第 1.3 节中的测试预言生成算法,对测试用例运行同一故障程序的运行结果是成功还是失败进行预测.

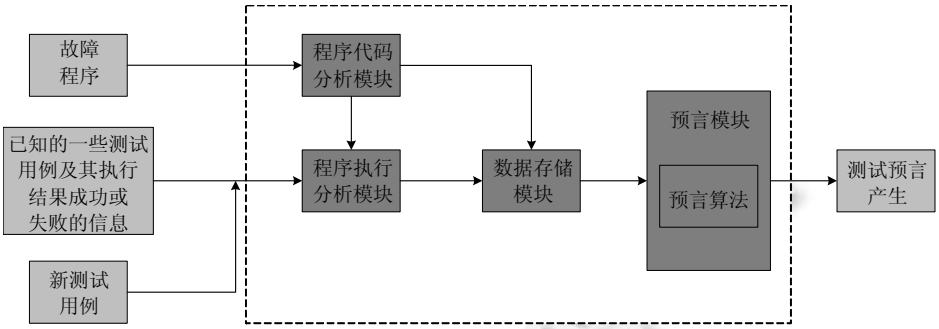

Fig.1 Assistant tool framework for experiments

图 1 实验工具总体架构

采用该实验辅助工具的具体实验步骤如下.

1) 将工具复制到需要进行测试预言生成的测试程序所在的文件夹中.
2) 在测试程序的文件夹中建立 output 文件夹,用于存放程序的执行覆盖文件 cov_result 和 mod_classify_result.
3) 将已知执行结果的所有测试用例存放在文件 trainset 中.
4) 运行数据收集程序 train.sh,输入参数为已知结果的测试用例存放的文件名、测试程序对象名、测试故障程序版本号.数据收集程序执行完成后,将收集的所有数据信息存放于数据库中.
5) 运行预测程序 predict.sh,计算新测试用例的测试预言.

3 实验结果及分析

3.1 实验对象的实验结果及分析

Tcas、print_tokens、schedule2、replace、tot_info、expressionParser 和 sort 共 129 个故障版本.为每一个故障版本设计 100 个新的测试用例(其中,成功和失败的比率与训练集一致),计算这 100 个测试用例预言正确的个数.图 2~图 8 分别展示了每类程序所有故障版本的预言正确数目的折线图.图中横轴表示每类程序的版本号,竖轴表示 100 个新测试用例使用本文的测试预言自动生成方法预测正确的个数.

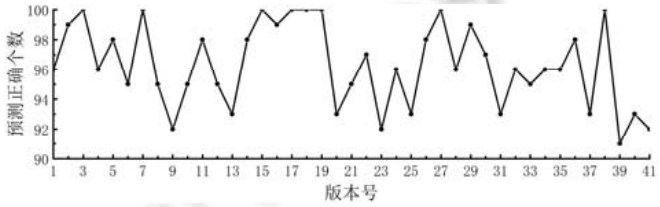

Fig.2 Test oracle prediction results of 41 tcas versions

图 2 Tcas 的 41 个版本的测试预言预测结果

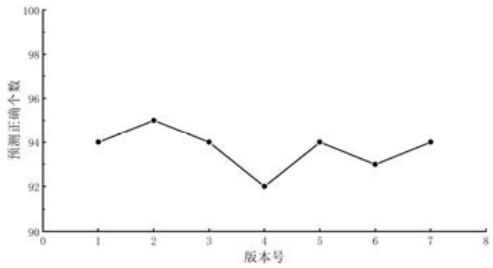

Fig.3 Test oracle prediction results of 7 print_tokens versions

图 3 Print_tokens 的 7 个版本的测试预言预测结果

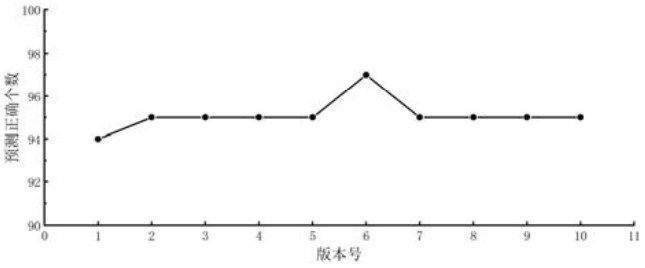

Fig.4 Test oracle prediction results of 10 schedule2 version

图 4 Schedule2 的 10 个版本的测试预言预测结果

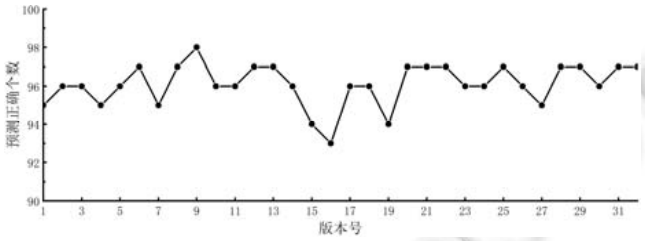

Fig.5 Test oracle prediction results of 32 replace versions

图 5 Replace 的 32 个版本的测试预言预测结果

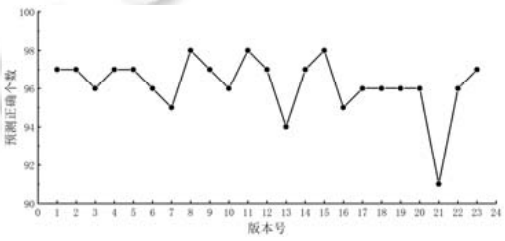

Fig.6 Test oracle prediction results of 23 tot_info versions

图 6 Tot_info 的 23 个版本的测试预言预测结果

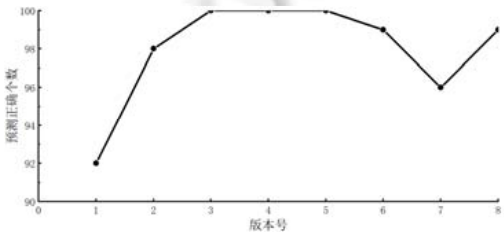

Fig.7 Test oracle prediction results of 8 expressionParser versions

图 7 ExpressionParser 8 个版本的测试预言预测结果

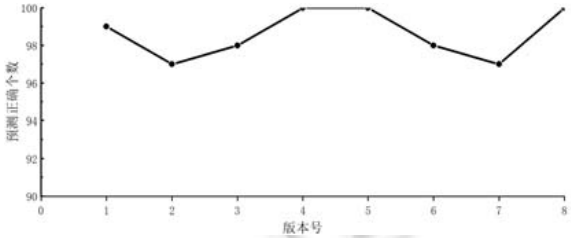

Fig.8 Test oracle prediction results of 8 sort versions

图 8 Sort 8 个版本的测试预言预测结果

程序 tcas 共有 41 个故障版本,每个版本运行 100 个新的测试用例,预测情况如图 2 所示.预测效果最差的是 V39,91 个测试用例预测正确.V3、V7 等 8 个故障版本的 100 个测试用例全部预测正确.41 个 tcas 版本预言生成的正确率平均为 96.3%.

程序 print_tokens 共有 7 个故障版本,每个版本运行 100 个新的测试用例,预测情况如图 3 所示.预测效果最差的是版本 V4,92 个测试用例预测正确.预测正确个数最多的是版本 V2,预测正确 95 个.7 个版本平均预言正确率为 93.7%.

程序 schedule2 共有 10 个故障版本,每个版本运行 100 个新的测试用例,预测情况如图 4 所示.故障程序 schedule2 的预测效果比较平均,有 8 个版本预测正确 95 个.10 个版本平均预言正确率为 95.1%.

程序 replace 共有 32 个故障版本,每个版本运行 100 个新的测试用例,预测情况如图 5 所示.预测效果最差

的是版本 V16,93 个测试用例预测正确.预测正确个数最多的是版本 V9,预测正确 98 个.32 个版本平均预言正确率为 96.1%.

程序 tot_info 共有 23 个故障版本,每个版本运行 100 个新的测试用例,预测情况如图 6 所示.预测效果最差的是版本 V21,91 个测试用例预测正确.预测正确个数最多的有版本 V8,V11 和 V15,预测正确 98 个.23 个版本平均预言正确率为 96.2%.

程序 expressionParser,sort 各有 8 个故障版本,每个版本运行 100 个新的测试用例,预测情况如图 7、图 8 所示.8 个版本平均预言正确率分别为 97.5%和 98.6%.

综上,对 129 个故障版本,每个版本预言 100 个测试用例,平均预言正确率为 96.2%,说明了本文给出的测试预言自动生成方法预测的有效性.

3.2 实验结果与其他相似方法的比较

从采用机器学习、现有测试用例的输入及执行结果协助自动生成新测试预言和完全自动化的角度来看,与本文相似的工作有文献[3–5]等.与他们相比,本文工作的优势如下.

1) 本文训练集数据量要求宽松,文献[5]要求失败的输入和成功的输入各占一半,而且数据量偏大,在实际应用中不可行.
2) 本文测试预言的生成应用范围更广.针对某个新的测试输入,上述 3 个文献生成的测试预言准确率完全依赖于训练集中的测试输入特征,在具体应用中具有一定的局限性和针对性;而本文的方法则是收集训练集中测试输入执行的语句覆盖、不同断点处的内存变量及其取值的集合等大量数据信息,更能真实反映被测试程序的行为,对于哪些输入数据个数少、输入取值有限或程序执行是否成功与输入的对应关系不明显等的应用程序而言,本文的方法皆可应用.
3) 本文的实证研究更深入和合理.上述文献的实证研究均通过个别案例程序并注入若干故障来检测方法的有效性.从故障版本的数量(共 129 版本)、故障类型的数量(比较和逻辑等各类运算符错误、常量赋值错误、变量定义错误、语句丢失、多余语句、变量赋值错误)和产生的测试预言数量角度,本文的实验更加深入和合理.
4) 测试预言的准确率方面本文有一定优势.从文献[5]的实验结果来看,对于案例 Tcas 而言,Sir 知识库中有效测试用例共 1 500 个,其中成功测试用例约占 85%;该文献在现有测试用例的基础上又构造了 1 500 个测试用例作为训练集,其预测准确率为 98.72%;但在 1 500 个测试数据集(还要求成功输入与失败输入数量相当)的情况下,其预测准确率约为 82%(文中未指明是否针对 Sir 知识库中 41 个版本都进行了实验,也未指明测试预言产生的数量).而本文针对 Tcas 的 41 个版本,每个版本均生成了 100 个测试输入的测试预言,平均预测准确率为 96.3%,其中,Tcas 的第 3、7、15、17~20 和 28 这 8 个版本的预测准确率为 100%.文献[5]中的其他实验对象,预测准确率最低为 84.66%.本文在其他实验对象中,在最少训练集合 1 052 个的情况下,预测准确率最低平均为 95.1%.

4 方法讨论

4.1 方法的实用性

本文提出的测试预言自动生成方法独立于源代码,不需要向源代码中插入任何断言;独立于编程语言,对任何一种编程语言都适用,没有构造形式模型的限制,具有很强的灵活性,适用范围广.本文的测试预言预测方法也适用于其他编程语言,例如 Java 和 C++语言的程序,但是本文搭建的实验环境仅支持 C 语言程序.

在软件调试过程中,程序员需要通过在程序关键语句处加断点,通过分析和比较部分断点处的变量取值以及语句执行序列等上下文来诊断故障,这是故障诊断的核心行为.所以本文假设符合实际情况,这一点在文献[6]的研究成果中也得到印证.

算法 1 生成的 t_{new} 测试预言的有效性与集合 T 和 St 密切相关,如果集合 T 或 St 的元素个数非常少,则算法

输出结果的准确率就会降低.该算法建立在已经存在的测试用例集合 T 之上,要求集合 T 存在一定数量.根据目前程序(C、C++、Java 等语言程序)的特点,仅有分支或 while 循环的判断条件可能导致测试用例执行的路径不同,所以在实际求解中,T 中与 t_{new} 最相似的测试用例集 St 也会满足算法求解的要求.

算法 1 的有效性并不依赖于 T 中测试用例的类型,并不要求 T 必须包含成功测试用例或必须包含失败测试用例.T 可以仅包含成功测试用例,也可以仅包含失败测试用例,也可以既包含成功测试用例又包含失败测试用例.所以本文训练集数据量要求宽松,并不要求失败的输入和成功的输入各占一半保持均衡.

本文测试预言生成算法在实际项目应用中可以迭代“滚雪球”式生成新的高质量测试用例.例如在项目真实的测试实践中,如果已经运行了 100 个测试用例都执行成功,那么对若干新的测试输入,可以用测试预言生成算法预测他们的输出成功还是失败,提示用户重点关注预测失败的新测试输入的执行.

4.2 测试预言生成的准确率问题

本文生成的某个新测试用例 t_{new} 的测试预言准确率与下述几个方面的要素密切相关.

1) 给定 t_{new},与 t_{new} 相似测试用例集合 St 相关,St 元素个数越多,训练集质量越高,预测结果趋于越精确.
2) 与各断点处敏感变量集的成功或失败的揭示能力相关,例如枚举变量、布尔变量及取值为离散值的变量,测试执行时,它们的取值较为集中而且重复率较高,所以这些变量一定被作为敏感变量求解出来,他们揭示程序成功或失败的能力更强.

本文通过分析不同版本的故障发现,如果被测试程序的故障语句是每个测试用例一定执行的语句,与 t_{new} 相似测试用例个数较多,我们的方法生成测试预言的准确率为 100%,例如 Tcas 的 V3、V7 等 V8 版本.如果被测试程序的故障语句是 if 或 while 中的语句,其对于测试输入集合而言,故障语句未必遇到每个测试用例都执行,故障语句执行次数越少,预测准确率就越低.主要原因是故障语句执行次数少,导致与预测测试用例高度相似的测试用例过少,并且故障敏感变量的影响指标(揭示成功或失败的能力)明显下降,例如 print_tokens 的故障版本 v4、tcas 的故障版本 v33 和 replace 的故障版本 v16 等,它们的故障语句都在多层嵌套的 if 判断条件中,通过其他条件不变,增加相似测试用例的控制实验,我们发现,如果增加相似测试用例集合中的个数,它们的预测结果会进一步提高.这也印证了与测试预言准确率相关的上述要素 1)和要素 2).

4.3 方法成本的评估

目前,本文方法未考虑多线程程序以及通过界面与用户交互的图形界面程序.测试套件下被测试程序各断点处变量值集合相关信息收集的工作量较大,它是测试预言生成算法的主要开销,也是本文的主要缺点和代价.

测试预言预测速度受断点个数 P、全部测试用例用例的个数 T 以及与待预测测试用例相似的测试用例个数 N 的影响.算法 1 计算相似测试用例集合的复杂度为 $O(T^2)$,每增加一个断,算法 1 就要在该断点处执行获取内存数据、计算敏感变量集和门限值等操作,因此算法 1 的效率与 P 呈线性关系.算法 2 比较相似测试用例集合中成功输入(令程序执行成功的输入)与失败输入(令程序执行失败的输入)的内存变量与其取值,计算敏感变量集合,该计算过程的复杂度为 $O(N^2)$.最后,算法 3 在每个断点处计算门限值,所以每次计算的复杂度是 $O(N^2)$.由于算法 1 调用了算法 2 和算法 3,综上所述,算法 1 的算法复杂度较大,为 $O(T^2+P\times N^2)$.

5 相关工作及结论

一些研究成果需要测试人员手动输入规格说明或模型,或训练神经网络的方法,辅助部分测试预言的自动生成.例如:(1) 在回归测试中,一些研究基于规格说明和组件应用编程接口规范,应用人工神经网络自动生成测试预言[7−10],这些方法需要手工构造规格说明;(2) 由测试者首先定义包含蜕变关系和数据挖掘算法的 JML 规格说明,然后采用蜕变测试技术来产生测试预言[11],中文参考文献[1],通过实验研究蜕变测试技术产生测试预言的有效性;(3) ASTOOT 通过检查两个不同的执行场景来产生测试预言,但是需要测试人员提供被测试系统的代数规格说明[12];(4) Li 等人提出了基于模型的测试预言生成策略,包括采用状态不变量作为基于状态图测试预言生成的依据[13];(5) Guo 等人首次提出一个通用的声明框架,该框架基于测试人员撰写的语义和控制流图

生成测试预言[14];(6) Wang 等人[15]利用支持向量机 SVM 作为监督的机器学习算法,测试人员采用智能测试预言库对被测试程序代码进行注释,根据函数调用收集测试跟踪信息,对训练集中每个测试轨迹提取特征作为 SVM 算法的输入,然后使用构建的 SVM 模型作为测试预言,通过两个案例程序的实验结果阐释了方法的有效性.

无需测试人员干预的测试预言自动生成方法.Arantes 等人采用逆向工程,通过静态和动态源代码分析的方式构造程序的一个结构化模型,然后基于该模型自动生成测试预言[3],提出的方法检测 4 了个案例中的部分故障.Goffi 等人借助现代软件系统,可以通过不同的执行顺序提供相同的功能的特征,通过检查给定执行序列与所有可用的冗余和假设等效的执行序列之间的等价性,设计和开发了一种完全自动化产生测试预言的技术,方法生成的测试预言发现了多达 53%的错误,并对开发人员撰写的 16 个案例中 4 个案例的测试预言,应用该方法提升了其有效性[4].Shahamiri 等人以被测试程序输入空间的行为作为训练集合,应用神经网络生成新测试集输入的测试预言;然后,该测试预言的预期结果与被测试程序真实的输出进行比对[16,17];最后,他们通过两个案例阐释其研究成果的有效性.由于被测试单元没有输出或输出形式的多样性,该方法以被测试单元的输出域作为准则产生测试预言有很大的局限性.Gholami 等人[5]以测试输入及其对应的成功或失败标记作为训练集,应用神经网络算法建立一个模型,为测试输入产生测试预言.他们通过两个案例研究阐释了其方法的有效性,但是该方法需要的训练集数量较大,而且要求失败的输入和成功的输入各占一半,而现实中积累大量执行失败的输入是不可能的.

本文工作属于无需测试人员干预的测试预言自动生成方法.本文采用监督的机器学习算法线性感知机作为测试预言自动生成的模型.首先,本文收集部分已知测试用例执行的语句覆盖和内存值集合,给出了程序在给定断点处的内存中敏感变量集求解算法;然后,将已知测试用例集合作为训练集,应用线性感知机求解每个断点处的门限值;最后,提出新测试用例的测试预言自动生成方法.

未来的工作将采用大型真实案例对本文提出的测试预言自动生成方法进行进一步分析与验证;同时,进一步探索测试预测的准确率与故障类型的关系.

References:

[1] Barr ET, Harman M, Mcminn P, Shahbaz M, Yoo S. The oracle problem in software testing: A survey. IEEE Trans. on Software Engineering, 2015,41(5):507−525. [doi: 10.1109/TSE.2014.2372785]

[2] http://sir.unl.edu/portal/index.php

[3] Arantes AO, de Santiago VA, Vijaykumar NL. On proposing a test oracle generator based on static and dynamic source code analysis. In: Proc. of the IEEE Int'l Conf. on Software Quality, Reliability and Security. IEEE, 2015. 144−152. [doi: 10.1109/QRS-C.2015.29]

[4] Goffi A. Automatic generation of cost-effective test oracles. In: Proc. of the ICSE 2014: Int'l Conf. on Software Engineering. 2014. 678−681. [doi: 10.1145/2591062.2591078]

[5] Gholami F, Attar N, Haghighi H, Asl MV, Valueian M, Mohamadyari S. A classifier-based test oracle for embedded software. In: Proc. of the 2018 Real-Time and Embedded Systems and Technologies (RTEST). 2018. 104−111. [doi: 10.1109/RTEST.2018.8397165]

[6] Zeller A. Yesterday, my program worked, today, it does not. Why? In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE'99). 1999. 253−267. [doi: 10.1145/318774.318946]

[7] Memon AM, Pollack ME, Soffa ML. Automated test oracles for GUIs. In: Proc. of the Century Applications. ACM Press, 2000. 30−39. [doi: 10.1145/357474.355050]

[8] Ye M, Feng BQ, Zhu L, Lin Y. Automated test oracle based on neural networks. In: Proc. of the IEEE Int'l Conf. on Cognitive Informatics. IEEE, 2007. 517−522. [doi: 10.1109/COGINF.2006.365539]

[9] Jin H, Wang Y, Chen NW, Gou ZJ, Wang S. Artificial neural network for automatic test oracles generation. In: Proc. of the Int'l Conf. on Computer Science and Software Engineering. IEEE Computer Society, 2008. 727−730. [doi: 10.1109/CSSE.2008.774]

[10] Xu WF, Ding T, Wang HL, Xu DX. Mining test oracles for test inputs generated from Java bytecode. In: Proc. of the Computer Software and Applications Conf. IEEE, 2013. 27−32. [doi: 10.1109/COMPSAC.2013.8]

[11] Murphy C, Shen K, Kaiser G. Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles. In: Proc. of the 2009 Int'l Conf. on Software Testing Verification and Validation (ICST 2009). Washington: IEEE Computer Society, 2009. [doi: 10.1109/ICST.2009.19]

[12] Doong RK, Frankl PG. The ASTOOT approach to testing object-oriented programs. ACM Trans. on Software Engineering and Methodology, 1994,3(2):101−130. [doi: 10.1145/192218.192221]

[13] Li N, Offutt J. Test oracle strategies for model-based testing. IEEE Trans. on Software Engineering, 2017,43(4):372−395. [doi: 10.1109/TSE. 2016.2597136]

[14] Guo HF. A semantic approach for automated test oracle generation. Computer Languages Systems & Structures, 2016,45:204−219. [doi: 10.1016/j.cl.2016.01.006]

[15] Wang F, Yao LW, Wu JH. Intelligent test oracle construction for reactive systems without explicit specifications. In: Proc. of the 2011 IEEE 9th Int'l Conf. on Dependable, Autonomic and Secure Computing (DASC). IEEE, 2011. 89−96. [doi: 10.1109/DASC. 2011.39]

[16] Shahamiri SR, Kadir WM, Ibrahim S, *et al*. An automated framework for software test oracle. Information and Software Technology, 2011,53(7):774−788. [doi: 10.1016/j.infsof.2011.02.006]

[17] Shahamiri SR, Wankadir WM, Ibrahim S, Hashim SZM. Artificial neural networks as multi-networks automated test oracle, Automated Software Engineering, 2012,19(3):303−334. [doi: 10.1007/s10515-011-0094-z]

马春燕(1978－),女,山东菏泽人,博士生,副教授,主要研究领域为软件工程,软件测试,故障定位,智能化软件开发.

张磊(1994－),男,硕士生,主要研究领域为软件工程,软件测试,故障定位,智能化软件开发.

李尚儒(1995－),男,硕士生,主要研究领域为软件工程,软件测试,故障定位,智能化软件开发.

张涛(1976－),男,博士生,副教授,主要研究领域为软件工程,软件测试,故障定位,智能化软件开发.

王慧朝(1992－),男,硕士生,主要研究领域为软件工程,软件测试,故障定位,智能化软件开发.