

# 基于数据流的测试用例自动生成研究

戴翔<sup>1</sup>, 毛宇光<sup>1,2</sup>, 吴非<sup>1</sup>, 薛一帆<sup>1</sup>

(1. 南京航空航天大学 计算机科学与技术学院, 江苏 南京 210016;

2. 南京大学 计算机软件新技术国家重点实验室, 江苏 南京 210093)

**摘要:**目前的数据流测试技术存在缺乏具体的数据驱动方法,测试用例的生成过程没有与测试需求结合起来等问题。基于此,文中提出一种自动生成测试用例的模型(TRGA),利用控制流图(CFG)计算生成测试用例所需的变量的定义使用对,使用数据类型结构图(DTG)来作为创建测试对象的数据驱动,利用遗传算法的搜索能力来生成数据,并提出了一种新的适应度计算方法。实验结果表明,该模型能够在减少搜索时间,降低生成测试用例规模的同时达到较高的测试覆盖率。

**关键词:**数据流;控制流图;数据类型结构图;适应度;测试用例

**中图分类号:**TP306

**文献标识码:**A

**文章编号:**1673-629X(2014)09-0001-05

doi:10.3969/j.issn.1673-629X.2014.09.001

## Research on Automatic Test Case Generation Based on Data Flow

DAI Xiang<sup>1</sup>, MAO Yu-guang<sup>1,2</sup>, WU Fei<sup>1</sup>, XUE Yi-fan<sup>1</sup>

(1. College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics,  
Nanjing 210016, China;

2. State Key Lab for Novel Software Technology, Nanjing University, Nanjing 210093, China)

**Abstract:** The current data flow testing techniques exist problems, for example, lack of specific data driven approach, test case generation process is not combined with the testing requirements and other issues. Based on this, propose a model for test cases of automatic generation, using the Control Flow Graph (CFG) to calculate the definition used pair for variable required by testing cases, using the Data Type Structure (DTG) as the data driving of establishing test object, applying genetic algorithms to generate test data, and present a new fitness calculation method. Experimental results show that the model can reduce the search time and reduce the size of generated test cases while achieving higher test coverage.

**Key words:** data flow; CFG; DTG; fitness; test case

## 0 引言

数据流测试是一种基于代码的白盒测试技术,已被广泛应用到面向对象的软件测试中。数据流测试利用程序中的数据流关系作为测试需求。Pande等提出了一种数据流中计算过程间的定义-使用对算法<sup>[1]</sup>。Harrold<sup>[2]</sup>等人提出利用CFG辅助数据流计算,并用计算的数据流关系指导测试用例的选择。虽然已有的随机技术能自动生成测试用例,但是其测试需求没有与测试过程结合起来,测试用例的生成具有盲目性。McCaffrey提出了能够产生测试数据的遗传算法模型<sup>[3]</sup>,Hermadi等<sup>[4]</sup>研究了遗传算法在测试数据产生问题上的可行性,并从参数的编码方法、适应度的构造、遗传算子的设计等方面进行研究。

文中提出了一种将数据流测试与遗传算法相结合的新模型。在这个模型中,采用构造方法控制流图和类的控制流图用于数据流的分析,构造方法序列生成器生成被测的方法序列,构造数据类型结构图用于指导创建被测的实例对象和驱动测试方法<sup>[5]</sup>。同时,还提出了将生成的定义-使用对作为两个目标函数来处理的适应度计算方法。该模型综合了数据流测试和遗传算法的优点,实现了代码覆盖和快速生成测试数据相结合,能够支持单一方法测试、交互方法测试、方法序列测试。

## 1 类的数据流测试

本质上来说,所谓的数据流测试就是对变量的定

收稿日期:2013-11-15

修回日期:2014-02-21

网络出版时间:2014-07-17

基金项目:国家自然科学基金资助项目(41301407)

作者简介:戴翔(1990-),男,硕士,研究方向为软件测试、数据仓库;毛宇光,副教授,研究方向为数据库系统、软件测试。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20140717.1228.020.html>

义和使用进行测试,利用产生的测试数据覆盖从变量的定义到变量的使用的路径<sup>[6]</sup>。程序中变量的使用有计算使用和断言使用两种。在计算或者输出语句中出现的使用就是计算使用,在断言中使用变量就是断言使用。将数据流计算得到的 def-use 对记为  $(d,u)$ , 一个  $(d,u)$  是一个有序对,表示语句  $d$  包含对变量  $v$  的定义,它可以通过某路径到达包含对变量  $v$  的使用的语句  $u$ 。

在下面的定义中,假设类  $C$  是一个待测的类, $d$  和  $u$  分别是包含对同一个变量的定义和使用的语句。

定义 1(单一方法内定义-使用对):设方法  $F$  是类  $C$  中的方法, $d$  和  $u$  分别是方法  $F$  中包含变量  $v$  的定义、使用的语句,则  $(d,u)$  是单一方法内定义-使用对。对此  $(d,u)$  的测试忽略方法调用所产生的数据流信息,成为单一方法测试。

定义 2(交互方法定义-使用对):设  $F_0$  是类  $C$  中的公有方法,而  $\{F_1, F_2, \dots, F_n\}$  是被  $F_0$  直接或间接调用的方法集。若有  $d$  是  $F_i$  中关于变量  $v$  的定义语句,  $u$  是  $F_j$  中对同一变量  $v$  的使用语句,并且  $F_i, F_j \in \{F_0, F_1, F_2, \dots, F_n\}$ , 则  $(d,u)$  是交互方法定义-使用对。对此  $(d,u)$  的测试称为交互方法测试。

定义 3(方法序列定义-使用对):设  $F_0$  是类  $C$  中的公有方法,  $\{F_1, F_2, \dots, F_n\}$  是被  $F_0$  直接或间接调用的方法,  $G_0$  是类  $C$  中的另一公有方法,  $\{G_1, G_2, \dots, G_n\}$  是被  $G_0$  直接或间接调用的方法。若有  $d$  是  $M_i$  中关于变量  $v$  的定义语句,  $M_i \in \{F_0, F_1, F_2, \dots, F_n\}$ ,  $u$  是  $G_j$  中关于同一变量  $v$  的使用语句,  $G_j \in \{G_0, G_1, G_2, \dots, G_n\}$ , 则  $(d,u)$  为方法序列定义-使用对。对此  $(d,u)$  的测试称为方法序列测试。

以类 Test 作为待测类,对实现测试用例的自动生成过程进行阐述。

## 2 基于数据流的测试用例自动生成框架

在面向对象软件中,存在着复杂的调用关系,增加了对类进行自动测试的难度。要实现对类的自动测试,要解决自动创建类的实例对象、自动生成被测方法序列、自动提供数据驱动被动方法、自动进行数据流的计算等问题。将实现测试自动化分为三个阶段:生成测试对象阶段、数据流计算阶段、实现自动测试阶段。以下为类 Test 源码。

```
public class Test {
    int a,b,c;
    public void operator() {
        int z,n;
```

```
1  int x = 2;
2  int y = 4;
3  if( x < 6) {
4      z = x;
        }
    else {
5      z = y;
        }
6  n = z;
7  while( n < 10) {
8      if( y > z) {
9      z = 2;
        } else {
10     n = n + z + 7;
        }
11     n = n + 1;
        }
12     System.out.println( x );
        }
    private void plus() {
13     a = a + 1;
14     b = b + 1;
15     c = c + 1;
        }
    public void afterOp() {
16     plus();
17     operator();
        }
        }
    public Test( int m, int n, int q ) {
18     a = m;
19     b = n;
20     c = q;
        }
```

### 2.1 生成测试对象

为了实现自动生成测试对象,需要了解方法之间的调用关系,完成方法序列的自动生成。文中提出利用方法调用图来存储方法调用关系,利用方法序列生成器来支持方法序列的自动生成。方法调用图(MCG)是一个有向图,利用其实现交互方法的测试,图中各节点代表方法,边表示方法之间的调用关系,用四元组  $(N,E,S,Z)$  描述,其中

(1)  $N$  是非空节点集合,包含入口节点、忽略节点。

(2)  $E$  是边的集合,即描述调用关系的集合。

(3)  $S \in N$ ,  $S$  是调用入口节点,指向类中的各个公有方法。

(4)  $Z \subset N$ ,  $Z$  为忽略节点集合,说明该方法没有产生调用关系。

根据以上描述,图1即为类Test的方法调用图。

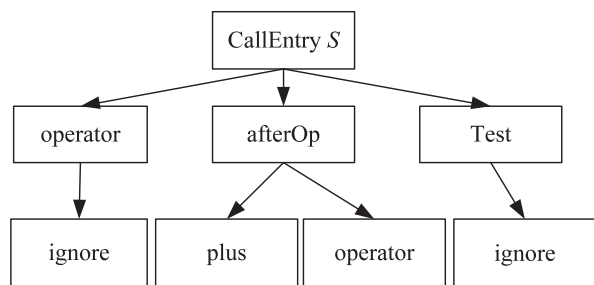


图1 类Test的方法调用图

在类测试中,由于方法执行的先后次序是动态决定的,因此必须首先得到类中的正确的方法序列。测试用例则围绕这些方法序列中的类成员变量的def-use对产生<sup>[7]</sup>。

利用Harrold和Rothermel<sup>[8]</sup>提出的构建CCFG的方法构建方法序列,类Test的方法序列如图2所示。

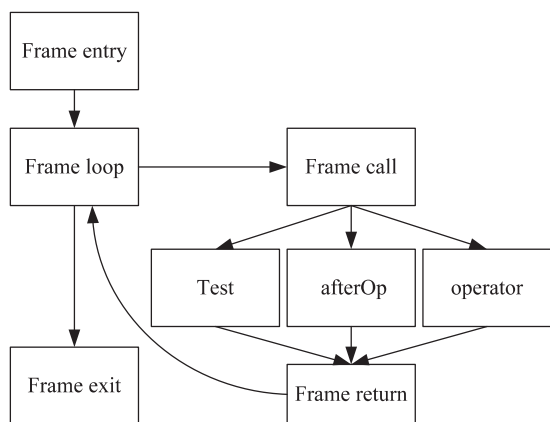


图2 类Test的方法序列图

为了实现自动创建测试对象的实例,必须要知道其构造方法和成员方法的参数结构,如参数的个数、顺序、类型等。对于非基本类型的参数,还要进一步确定该参数的构造方法,从而形成一个层次结构。使用DTG来描述参数的层次结构。DTG可用五元组 $(N, E, S, T, P)$ 来描述,其中

(1)  $N$ 为可空节点集合,每个节点代表一个参数的数据类型。

(2)  $E$ 为边集合,对于任意的 $e \in E$ , $e$ 为有序偶 $\langle t_i, t_j \rangle$ ,并且 $t_i, t_j \in N$ ,表示 $t_i$ 引用了 $t_j$ 。

(3)  $S \in N$ ,为入口节点,以方法名标记。

(4)  $T$ 为标签集合,标签代表参数名。

(5)  $P$ 为 $E \rightarrow T$ ,是边与标签之间的关联映射。

在知道所需的参数结构后,方法序列就可以结合MCG和DTG自动进行测试对象的创建和驱动测试方法。

## 2.2 数据流计算阶段

数据流计算阶段是整个测试用例自动生成的核心步骤,利用CFG计算得到一系列 $(d, u)$ 对作为测试需

求。控制流图 $G = (V, E)$ 是一个有向图,其中 $V$ 是一个非空节点集,包含特殊的entry入口节点和exit出口节点以及函数中的所有节点; $E$ 是控制流边组成的集合,每条边是从一个节点到另外一个节点的控制转移的有序对。图3所示为Test类中方法operator的控制流图。

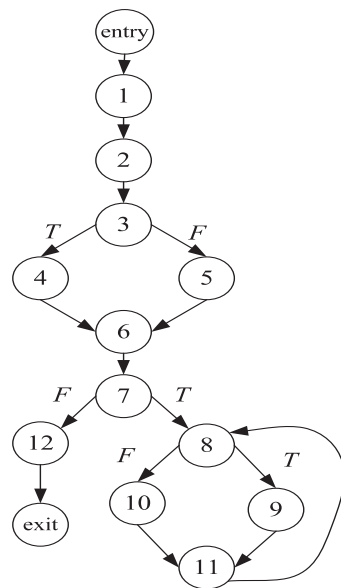


图3 方法operator的控制流图

利用方法的CFG可以完成对单一方法的测试。当需要对交互方法、方法序列进行测试的时候,还需要类中方法之间的调用关系和方法序列的数据流向,故需要构造类控制流图。类控制流图是将方法调用加入到方法序列生成器框架,同时将方法调用图中的方法节点替换成为方法控制流图。得到类控制流图后,进行数据流计算,得到各个变量的dcu。

## 2.3 实现自动测试阶段

在创建完测试对象的实例,得到完整的数据流关系后,就可以由遗传算法自动生成测试数据<sup>[9-10]</sup>。

不同于随机算法(RA)的任意产生测试数据,数据遗传算法是目标导向性搜索方法<sup>[11]</sup>,它可以在当前种群中,不断搜索适应度较高的个体,通过一系列选择、交叉、变异来生成覆盖测试需求的测试数据。

### 2.3.1 编码

传统的遗传算法编码有二进制编码、符号编码、浮点数等。文中采用Michalewicz提出的编码规则,来对变量的值进行编码。假设待测程序中有 $n$ 个输入变量 $x_1, x_2, \dots, x_n$ ,每个变量 $x_i$ 的取值范围是 $[a_i, b_i]$ 。 $d$ 表示变量 $x_i$ 的小数位数。为了达到这个精度,需将取值范围划分成 $(b_i - a_i) \cdot 10^d$ 个相同大小的区间。假设存在 $m$ 使得 $(b_i - a_i) \cdot 10^d \leq 2^m - 1$ ,则可以用长度为 $m$ 的二进制串 $b_m b_{m-1} b_{m-2} \dots b_2 b_1$ 表示变量 $x_i$ 。由变量的编码推出解码公式为

$$x = a_i + \left( \sum_{i=1}^m b_i \cdot 2^{i-1} \right) \cdot \frac{b_i - a_i}{2^m - 1} \quad (1)$$

### 2.3.2 适应度函数的构造

为了使遗传算法生成的测试数据能覆盖指定的路径<sup>[12-14]</sup>,所以要衡量每个个体在覆盖特定路径的优劣程度。如果生成的测试数据,能够较好地覆盖指定路径,则其适应度越好,反之则越低。

Girgis 提出用已覆盖的 def-use 路径个数与总的 def-use 路径个数的比值作为适应度函数,但该方法不能度量测试用例之间的相似度。为了解决这个问题,将 def-use 对当成两个目标函数值来处理,用测试数据实际覆盖的路径与该节点的支配路径进行比较。

对于有向图  $G = (V, E)$ , 如果从入口节点  $n_0$  到节点  $m$  的任一路径都需要经过节点  $n$ , 那么就称  $n$  支配了  $m$ 。由  $n_0$  到  $n$  所必须经过的其他节点组成的路径称为节点  $n$  的支配路径, 记为  $\text{dom}(n)$ 。

对于定义使用对  $(d, u, \text{var})$ , 其中  $d$  和  $u$  分别是  $\text{var}$  的定义和使用节点, 假设当前种群中有一变量为  $v$ , 其适应度函数  $\text{ft}(d, u, v_i)$  的计算方法如下:

(1) 利用支配树找出定义节点和使用节点的支配路径  $\text{dom}(d)$ ,  $\text{dom}(u)$ ;

(2) 增加定义节点到  $\text{dom}(u)$ , 确保变量  $v$  在使用前能被定义到;

(3) 解码当前变量  $v$ , 执行程序, 记录被覆盖到的  $\text{dom}(d)$ ,  $\text{dom}(u)$ , 记为  $\text{cdom}(d)$  和  $\text{cdom}(u)$ ;

(4) 统计没有一次被覆盖到的  $\text{dom}(d)$  和  $\text{dom}(u)$ , 记为  $\text{udom}(d \vee u)$ ;

(5) 统计被覆盖到的次数大于 1 的  $\text{dom}(d)$  和  $\text{dom}(u)$ , 记为  $\text{fdom}(d \vee u)$ 。

于是得出适应度函数为

$$\text{ft}(d, u, v_i) = \frac{1}{2} \times \left( \frac{|\text{cdom}(d)|}{|\text{dom}(d)|} + \frac{|\text{cdom}(u)|}{|\text{dom}(u)|} \right) \quad (2)$$

同时, 基于假设

(1) 能覆盖定义节点的测试数据, 比没有覆盖任何节点的测试数据具有更高的相似度;

(2) 没有覆盖一个特定的节点并且试图再次覆盖这个目标节点的测试数据, 比没有覆盖这个特定的节点并且不再尝试进行覆盖的测试数据具有更高的相似度。

提出测试数据的相似度 (DS):

$$\text{DS} = \frac{1}{2} \times \min \left( 0.9, \frac{|\text{fdom}(d \vee u)| - |\text{udom}(d \vee u)|}{|\text{dom}(d)| \times |\text{dom}(u)|} \right) \quad (3)$$

结合测试数据的相似度, 故适应度函数为

$$\text{Ft}(d, u, v_i) = \text{ft}(d, u, v_i) + \text{DS} \quad (4)$$

## 2.4 基于数据流的测试用例自动生成算法 (TRGA)

变量声明:

TR: 测试需求 (定义-使用对);

TC: 测试数据;

CurrentPop: 当前测试数据;

NewPop: 下一代测试数据;

TRCovered: 已覆盖的测试需求。

输入:

P: 待测程序;

TestReqAll: 所有测试需求;

InitPop: 初始测试数据。

输出: 被覆盖到的测试需求和测试数据。

开始:

(1) 按照编码规则, 初始化 InitPop;

(2) 依次将 InitPop 中的每个测试数据作为 P 的输入, 并执行 P;

(3) 检查 TestReqAll 中的 TR 是否有被覆盖的, 若有则标记为已覆盖;

(4) 将标记的 TR 加入到 TRCovered, 并从 TestReqAll 中删除相应的 TR;

(5) 更新当前测试数据,  $\text{CurrentPop} = \text{InitPop}$ ;

(6) 若 TestReqAll 中仍有 TR 没有被覆盖到并且未达到最大遗传代数, 则执行步骤 (7) ~ (9); 否则跳转到步骤 (10);

(7) 按照给出的适应度函数评价 TC;

(8) 进行遗传算法中的选择、交叉、变异操作得到 NewPop;

(9)  $\text{InitPop} = \text{NewPop}$ , 重复步骤 (2) ~ (6);

(10) 测试结束。

## 3 实验分析

采用 3 个 Java 程序, 分别是找到数据最大值、三角形分类、求三角形面积, 对其进行实验分析。程序的详细信息如下:

程序 1: 求数据的最大值, 共 55 行, 复杂度低, def-use 数为 53;

程序 2: 判别三角形形状, 共 61 行, 复杂度中, def-use 数为 174;

程序 3: 求三角形的面积并求最大值, 共 143 行, 复杂度高, def-use 数为 281。

对遗传算法的参数进行设置, 最大遗传代数为 200, 初始种群规模为 10, 选择和遗传概率为 0.80, 变异概率为 0.15。分别采用 TRGA 和 RA 算法对三个程序进行 all-use 测试, 并从覆盖率、生成的测试用例数、搜索时间三个方面将两种算法进行比较。

图4表示的是按照 all-use 覆盖标准,两种算法所得到的覆盖率。例如,对于程序1而言,TRGA 对所有的  $(d,u)$  对获得 100% 的覆盖率,而 RA 的覆盖率为 89.23%。

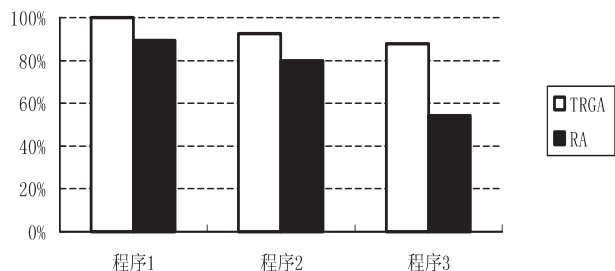


图4 覆盖率比较

图5为满足 all-use 覆盖标准的两种算法所产生的测试用例的数目。例如,对程序2,为覆盖所有的  $(d,u)$  对,TRGA 产生了 683 个测试用例,而 RA 产生了 1 037 个测试用例。

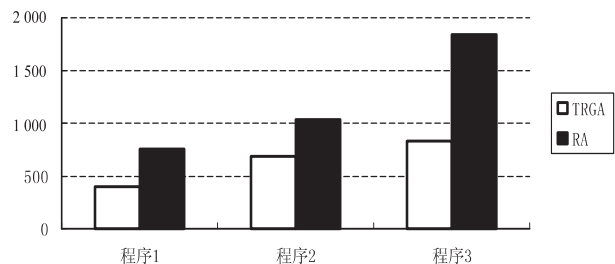


图5 测试用例数比较

图6为满足 all-use 覆盖标准的两种算法所需要的搜索时间。例如,对程序3而言,采用 TRGA 算法需要 163 s,而 RA 算法则需要 358 s。

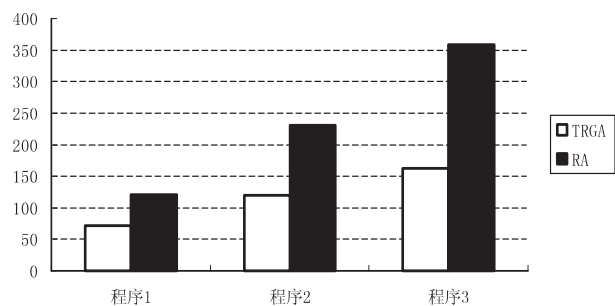


图6 搜索时间比较

实验结果表明,TRGA 可以减少搜索时间,降低生成用例的规模,同时获得对测试需求较高的覆盖率。

## 4 结束语

测试用例的自动生成其实就是优化问题,文中提出了一种基于数据流的用例自动生成的框架,利用数据流关系作为测试需求、数据类型结构图作为数据驱

动,并结合遗传算法对测试生成的数据的评价能力,从而将数据流技术与遗传算法结合在一起,提高了测试用例自动生成的效率。

文中缺乏对复杂程序结构的用例自动生成的研究,如循环、嵌套等,这将是今后进一步研究的方向。

## 参考文献:

- [1] Pande H, Landi W, Ryder B G. Interprocedural def-use associations in C programs[J]. IEEE Transactions on Software Engineering, 1994, 20(5): 385-403.
- [2] Ghiduk A S, Harrold M J, Girgis M R. Using genetic algorithms to aid test-data generation for data-flow coverage[C]//Proc of 14th Asia-Pacific software engineering conference. Aichi: IEEE, 2007: 41-48.
- [3] McCaffrey J D. Generation of pairwise test sets using a genetic algorithm[C]//Proc of 33rd annual IEEE international conference on computer software and applications. Seattle: IEEE, 2009: 626-631.
- [4] Hermadi I, Lokan C, Sarker R. Genetic algorithm based path testing: challenges and key parameters[C]//Proceedings of 2010 second world congress on software engineering. Wuhan: IEEE, 2010: 241-244.
- [5] Gupta N, Mathur A P, Soffa M L. Automated test data generation using an iterative relaxation method[J]. ACM SIGSOFT Software Engineering Notes, 1998, 23(6): 231-244.
- [6] 许力, 陈江勇. 基于遗传算法的数据流测试用例自适应生成算法[J]. 计算机系统应用, 2013, 22(7): 90-94.
- [7] 张雪萍, 范中山, 王家耀, 等. 基于数据流的类测试技术研究[J]. 计算机工程与应用, 2005, 41(9): 40-42.
- [8] Harrold M J, Rothermel G. Performing data flow testing on classes[J]. ACM SIGSOFT Software Engineering Notes, 1994, 19(5): 154-163.
- [9] Wang Xibo, Su Na. Automatic test data generation for path testing using genetic algorithms[C]//Proc of third international conference on measuring technology and mechatronics automation. Shanghai: IEEE, 2011: 596-599.
- [10] 赵明, 张毅坤, 沈建雄, 等. 基于遗传算法的测试用例生成工具研究[J]. 计算机工程, 2005, 31(13): 151-153.
- [11] 周献中, 孙勇成, 江金龙. 基于使用模型和遗传算法的测试数据自动产生技术[J]. 兵工学报, 2006, 27(6): 1051-1055.
- [12] 张磊, 王晓军. 基于遗传算法的业务流程测试[J]. 计算机技术与发展, 2010, 20(3): 155-158.
- [13] 史娇娇, 姜淑娟. 基于遗传算法的动态可变参数的测试数据自动生成工具[J]. 计算机科学, 2012, 39(5): 124-127.
- [14] 白凯, 崔冬华. 基于遗传算法的测试数据自动生成[J]. 计算机与数字工程, 2010, 38(2): 52-54.

基于数据流的测试用例自动生成研究

作者:

[戴翔](#), [毛宇光](#), [吴非](#), [薛一帆](#), [DAI Xiang](#), [MAO Yu-guang](#), [WU Fei](#), [XUE Yi-fan](#)

作者单位:

[戴翔, 吴非, 薛一帆, DAI Xiang, WU Fei, XUE Yi-fan\(南京航空航天大学 计算机科学与技术学院, 江苏 南京, 210016\), \[毛宇光, MAO Yu-guang\\(南京航空航天大学 计算机科学与技术学院, 江苏 南京 210016; 南京大学 计算机软件新技术国家重点实验室, 江苏 南京 210093\\)\]\(#\)](#)

刊名:

[计算机技术与发展](#)

英文刊名:

[Computer Technology and Development](#)

年, 卷(期):

2014 (9)

本文链接: [http://d.wanfangdata.com.cn/Periodical\\_wjz201409001.aspx](http://d.wanfangdata.com.cn/Periodical_wjz201409001.aspx)