2018年2月 西北工业大学学报 Feb. 2018
第36卷第1期 Journal of Northwestern Polytechnical University Vol.36 No.1

DOI：10.1051/jnwpu/20183610156

基于JDart的测试用例自动生成与优化

吴潇雪[1]，郑炜[2]，王培源[2]，王培甲[2]，樊宋宇[2]

（1.西北工业大学 自动化学院，陕西 西安 710072；2.西北工业大学 软件与微电子学院，陕西 西安 710072）

摘 要：基于符号执行的测试用例生成方法，以其高可靠性得到了学术界和工业界广泛关注。然而，已有工具大都面向C或者C++程序，面向Java的符号执行工具发展相对较慢。JDart是表现较好的一款开源的面向Java的符号执行工具，但是对复杂数据类型比较数组却支持很弱，因此，在对JDart工具以及动态符号技术进行分析的基础上，通过对JDart测试用例生成能力和存在问题的深入剖析，针对数组处理进行改进，以提高生成测试用例的代码覆盖率，保证测试质量。最后，通过用三角形程序实例进行验证，结果表明，改进后的JDart工具能够完全探索函数中关于数组处理的所有路径。

关 键 词：测试用例生成；JDart；符号执行；优化策略

中图分类号：TP311.5 **文献标志码**：A **文章编号**：1000-2758(2018)01-0156-06

作为软件测试工作重要内容的测试用例生成，是影响软件测试效果的重要环节，测试用例的自动产生，对保证测试质量、提高测试效率至关重要[1-2]。其中，符号执行引起了广泛关注[3-5]，并产生许多开源的符号执行工具[6]，诸如CUTE/jCUTE、CREST、JPathFinder[7]、KLEE[8]、SPF以及JDart[9-10]等。面向C的符号执行工具，大多依赖于较为成熟的LLVM平台和KLEE，发展较为迅速，在工业界也得到广泛应用。而面向Java语言的符号执行技术则发展相对较慢，尚处于探索阶段。可靠且高效的面向Java程序的测试用例生成技术和工具亟待开发。

JDart[10]是一款较为成功的面向JAVA的符号执行工具，它提供了良好的模型和框架。然而，目前JDart只能支持基本简单的数据类型，如Int、Char的符号化，而对于复杂数据类型如Array、String、Object等还无法进行符号化，因而影响了其推广和应用。

本文根据对JDart工具及动态符号技术的研究和应用，通过对JDart测试用例生成能力和所存在问题进行了深入剖析，针对数组符号化和约束求解，给出改进和优化策略，以提高生成测试用例的覆盖率，保证测试工作的质量。

1 基于符号执行的测试用例生成

符号执行是一种可靠、经典的白盒测试用例生成方法。其基本原理是将程序中变量值用符号标记替换，并以符号值模拟程序执行的过程。基于符号执行的测试用例生成方法，主要通过计算每条符号执行路径所需要满足的约束条件，来得到测试输入。

1.1 相关定义

为了便于讨论，我首先列出与符号执行测试用例生成相关的定义[11-12]。

定义1 待测系统SUT(system under test)。给定n个称之为待测软件的参数$p_i(i=1,2,\cdots,n)$。这些参数可能代表系统的配置参数、内部实践或者用户输入。

定义2 路径条件PC(path condition)。汇积了执行某条路径中每一条语句必须满足的约束，不可达路径的执行路径定义为P_U。

定义3 （约束集C）设由对源程序进行符号执行所得到的路径集合为$\boldsymbol{C}=\{c_1,\cdots c_i,\cdots,c_n\}$，则称$\boldsymbol{C}$为约束条件集合，其中$c_n$表示某个具体的路径条件表达式，$n$为约束条件的总数。

收稿日期：2017-05-08 **基金项目**：国家自然科学基金(61402370)与航天科技基金(HT-2017-XG)资助

作者简介：吴潇雪(1983—)，女，西北工业大学博士研究生，主要从事软件故障检测与云环境安全研究。

定义 4 （约束 PC）对关于数值的路径条件,定义为数值约束 N_{PC};对关于字符串的约束,定义为字符串约束 S_{PC};对 PC 中既包含数值约束又包含字符串约束,则称该 PC 为混合约束条件 H_{PC}。

定义 5 （测试输入 T）由 N_{PC} 求解所得到的值称为关于数值约束条件的测试输入,定义为 T_N;由 S_{PC} 求解所得到的值称为关于字符串约束条件的测试输入,定义为 T_S;相应的 T_N 和 T_S 的集合即为关于该执行路径的测试输入 $\boldsymbol{T}$。

定义 6 （求解过程 $\boldsymbol{S}$）将对约束条件结合进行求解的过程记为 $\boldsymbol{S}$,其中 $\boldsymbol{S}$ 的输入是约束条件集合,输出为测试输入集合与不可达路径条件,可写成 $S(C) = \boldsymbol{T}$ AND $\boldsymbol{P}_U$,式中 $\boldsymbol{C} = \{c_1,\cdots c_i,\cdots,c_n\}$;$\boldsymbol{T} = \{T_1,\cdots T_i,\cdots,T_N\}$。

1.2 测试生成过程

符号执行的核心思想,是用抽象化的符号表示程序中的具体输入变量;通过解析程序的语句,为每条路径生成路径表达式,来生成符号执行树,并通过对路径表达式进行求解获得符号变量解的集合[7]。例如对于图 1 所示的待测示例代码,通过符号执行分析,可以得到符号执行语法树如图 2 所示。

```
1    int x,y;
2    if(x>y){
3      x=x+y
4      y=x-y;
5      x=x-y;
6      if(x-y>0)
7           Assert false;
8      }
9      print(x,y)
```

图 1　示例代码

通过该语法分析树,可以得到程序执行路径、路径约束条件以及对应的输入,如表 1 所示。这就是一个简单的基于符号执行的测试用例生成过程。

表 1　路径条件、路径、以及生成输入

Path	Path Condition	Program Input
1,8	$X<=Y$	$X=1$　$Y=1$
1,2,3,4,5,7	$X>Y$ &$Y-X<=0$	$X=2$　$Y=1$
1,2,3,4,5,6	$X>Y$ & $Y-X>0$	NONE

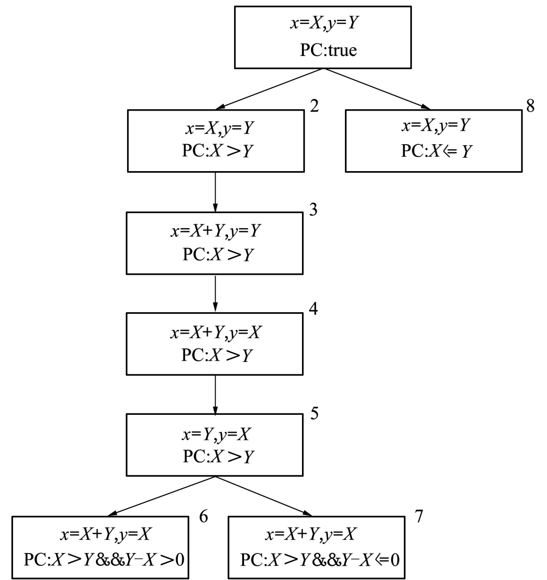

图 2　符号执行树

2 基于 JDart 的测试用例生成

2.1 JDart 工具简介

JDart[7]是一个面向 Java 的动态符号分析框架,采用模块化架构。其中,执行动态探索的主要组件与有效构建约束的组件进行通信,并与约束求解器进行接口通信。这些组件可以轻松扩展或修改,以支持多个约束求解器或不同的探索策略。此外,连同其最近的开源,使 JDart 成为研究和实验的理想平台。在当前版本中,JDart 支持 CORAL,SMTInterpol 和 Z3 求解器,并能处理包含位操作、浮点算术和复杂算术运算(例如,三角和非线性计算)的约束求解器。

2.2 JDart 测试用例生成机制

用动态符号执行工具 JDart 测试程序时,还需要 2 个辅助工具:JConstraints 和 JConstraints-Z3。符号执行工具在执行过程中需要使用约束求解器对收集到的约束条件求解,JDart 中常用的约束求解器是 Z3,而 Z3 求解器为了提高求解的效率,对常用的数据类型定义了独特的数据结构变量。因此,想要使用 Z3 求解器,需要使用 Z3 求解器提供的 API 接口,将收集到的约束条件中的变量和常量转化为 Z3 求解器中的数据形式。动态符号执行工具 Jdart 的工作流程如下:

Step1　初始化被测程序代码模块,创建一个新的任务。

Step2　在类 ConcolicMethodExplorer 的 initializeMethod( ) 函数中,调用函数 initializeSymbolicParams( ),对测试方法输入值符号化。

Step3　创建符号值。对于基本数据类型,直接创建符号值。对于对象或数组等复杂数据类型时,调用 processObject( )方法进行处理,在 processObject( )方法中,进行符号值创建。

Step4　收集约束条件并进行约束求解。收集被测程序中的分支条件作为路径约束,将路径约束添加到约束集合中,按深度优先路径调度策略对约束集合中的某一项或几项取反,得到路径树上下一条路径的约束集合,然后调用 JConstraints-Z3 中的 add( )方法,将路径约束转化为 Z3 求解器可以识别的约束条件,并将其添加到约束求解器中,最后调用 solve( )函数对约束集合求解,得到下一条路径的输入值。并依次执行,直到遍历被测程序所有路径的执行。

2.3　存在问题

JDart 实现了基本的测试用例生成思路和框架。但是,对于许多具体的数据类型(如对象、数组、字符串等)还无法支持。图 3 是一个简单的包含数组的程序,图 4 展示该程序预期的执行路径。

```
1   int ArrayTest(int[ ]a)
2   {
3     if(a = =null) {
4       return 0;
5     }else if(a.length>0) {
6         if(a[0] = =123) {
7           throw new Exception("bug");
8         }
9         return a[0];
10      } else {
11          return 0;
12        }
13  }
```

图 3　测试程序代码

理论上,动态符号执行测试时会遍历程序所有 4 条路径,如图 4 所示。但是, JDart 根据初始输入数组值的不同,却只能探索到 1 到 2 条路径,例如:初始输入数组为 null,只能探索到路径:3->4;如果数组为{1,2,3},则可以探索到 2 条路径:3->5->6->9 和 3->5->6->7。造成这种情况的原因在于:JDart 在数组符号化后,具体执行程序一次,收集该条路径的约束条件时,会检测到数组 a 不是符号值,

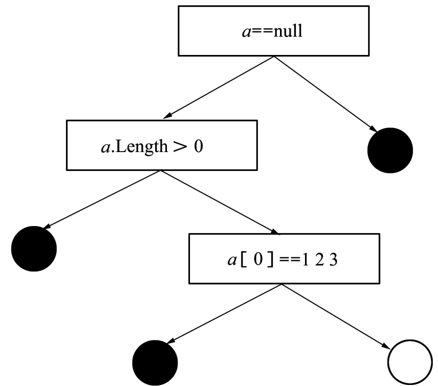

图 4　预期路径

不能对路径条件 a= =null 和 a.length> 0 取反,从而导致 JDart 执行时对这种路径分支只执行了一条路径。因此,应当对 JDart 处理数组的符号化时所存在的问题进行分析和改进,使 Jdart 测试程序时可以实现如图 4 所示的数组类型程序代码,遍历被测程序的所有路径。

3　分析及优化

3.1　问题分析

对 JDart 的程序代码进行分析,发现当被测程序的初始输入为数组时,JDart 对数组的符号化是将数组视为一个字节数组,然后将数组中每个字节与一个不同的符号量相关联实现数组的符号化的。在上一节介绍了 JDart 的执行过程,在对测试方法的初始输入符号化时,当输入值为对象或数组,调用 processObject( )方法进行处理,processObject( )中调用的 doProcessObject 方法会根据对象类型的不同,调用不同的处理器,来决定是否符号化和如何符号化。

JDart 根据初始输入数组值的长度对数组符号化,但数组符号化是定长的。例如:数组 a 的初始输入长度为 10,那么 JDart 就会创建 10 个符号变量 a[0],a[1],…, a[9]。在此情况下,在测试程序执行过程中,JDart 所收集的约束条件中的数组元素是有符号值的,但是,数组本身和数组长度没有符号值,对包含数组本身和数组长度的约束条件部分,JDart 将只能执行一条路径。

3.2 改进策略

只创建一个数组本身的符号变量 a,然后根据数组的数据结构性质,用函数表达式来表示长度和下标。这样,当 JDart 在动态符号执行被测程序时,若遇到涉及数组下标和长度的分支,就可以收集到这些路径的约束条件,再通过对路径约束取反,就能实现动态符号执行工具 JDart 对这些分支条件中 2 条分支的遍历,从而可以提高 JDart 对被测程序的覆盖率,使错误检测模块尽量检测到程序中所有可能的 bug。

综上所述,修改步骤可描述如下:

Step1 在 JConstraints 中,添加对数组的支持,将 Z3 求解器求解出的 model 中,涉及数组的模块解析成数组形式输出,使 JDart 中可以创建对应的数组类型符号值。

Step2 在 JConstraints-Z3 中,添加对数组的支持。JConstraints-Z3 的功能主要有 3 点:①将传入的约束表达式,转化为 Z3 求解器可以识别的 Z3 断言形式;②调用 Z3 求解器提供的 API,对转化后的 Z3 断言求解;③对求解出的 model 进行遍历,将求解的结果转化为用户常见的格式。因为 Z3 求解器能支持对数组的求解,因此,仅需针对第一点和第三点进行修改。

Step3 对 JDart 的数组符号化进行优化。对数组符号化的优化,主要为了使以下 3 个涉及数组的分支条件覆盖到:$a==null$、$a.length>0$ 和 $a[i]==x$。因此,仅从 3 个方面实现对数组符号化的优化:①在类 PrimitiveArrayHandler 的 annotateObject 方法中,直接创建数组类型符号;②对数组长度的读取,修改 arraylength 指令,在 gov.nasa.jpf.jdart.bytecode 下新建一个 ARRAYLENGTH 类,用以继承 jpf-core 中重写的 ARRAYLENGTH 类,如果数组是符号值,则生成 length 表达式;③对数组元素的读取,可修改 iaload 指令,同样新建一个 IALOAD 类,来生成 item 表达式。

4 结果验证

下面采用经典的三角形程序,对本文上述对数组符号化改进的效果进行验证。三角形代码如图 5 所示。

```
1  public class Input {
2
3    public static void main(String[]args) {
4      Input localTest=new Input();
5      int[]a={2,3,3};
6      localTest.isConstructiveTriangles(a);
7    }
8
9    private void isConstructiveTriangles(int[]a) {
10       if (a[0]<a[1]+a[2] && a[1]<a[0]+a[2] && a[2]<a
   [1]+a[0]) {
11         assert true;
12       } else {
13         assert false;
14       }
15     }
16 }
```

图 5 判定输入数组是否构成三角形代码

对上述代码,采用改进前后的 JDart 程序分别执行。表 2 和表 3 为 2 次执行遍历路径所对应的测试用例,其中 PC 为每条路径的约束,Program Input 为该路径所对应的测试输入。结果表明,改进之前的 JDart,对图 8 所示的代码可以探测到 4 条路径。而改进之后,则可探测到 10 条路径,即该函数的所有执行路径。因此,实现数组符号化后,JDart 对包含数组的被测程序,路径覆盖率有显著提高,不仅实现了对数组为空、长度和下标的判断,并且对数组的长度添加了约束,以防止数组长度过大导致 Z3 求解出现异常。

表 2 JDart 优化前生成的测试用例

序号	PC	Program Input
1	$a[0]>=a[1]+a[2]$	$a[0]=0$, $a[1]=-1\ 250\ 829\ 122$, $a[2]=667\ 999\ 421$
2	$a[0]<a[1]+a[2]$ && $a[1]>=a[0]+a[2]$	$a[0]=-536\ 870\ 916$, $a[1]=-3$, $a[2]=536\ 870\ 913$
3	$a[0]<a[1]+a[2]$&& $a[1]<a[0]+a[2]$&& $a[2]>=a[1]+a[0]$	$a[0]=-2$, $a[1]=-1$, $a[2]=536\ 870\ 912$
4	$a[0]<a[1]+a[2]$&& $a[1]<a[0]+a[2]$&& $a[2]<a[1]+a[0]$	$a[0]=2$, $a[1]=3$, $a[2]=3$

表3 JDart 优化后生成的测试用例

序号	PC	Program Inpt	序号	PC	Program Inpt
1	a.ref <=0	a=null	7	a.ref>0 && a[0]>0 &&a[1]>0 &&a[2]>0 && 2<a.length<=100 &&a[0]>=a[1]+a[2]	a={2 145 681 341, 1 069 547 450, 1 076 101 125}
2	a.ref>0 && a[0]<=0	a={-2 147 483 646}			
3	a.ref>0 && a[0]>0 && a.length<=1	a={2 145 681 341}			
4	a.ref> 0&& a[0]>0 && a[1]<=0 && 1<a.length<=100	a={2 145 681 341, -2 147 483 136}	8	a.ref>0 && a[0]>0 && a[1]>0 &&a[2]>0 && 2<a.length<=100 &&a[0]<a[1]+a[2] && a[1]>=a[0]+a[2]	a={2 141 487 037, 1 069 547 450, 1 076 101 124}
5	a.ref> 0 && a[0]>0 &&a[1]> 0 && a.length<=2	a={2 145 681 341, 1 069 547 450}	9	a.ref>0 && a[0]>0 &&a[1]>0 &&a[2]> 0 && 2<a.length<=100 &&a[0]<a[1]+a[2] && a[1]<a[0]+a[2] && a[2]>=a[0]+a[1]	a={2 141 487 037, 2 143 289 274, 2 359 300}
6	a.ref> 0 && a[0]>0 && a[1]>0 && a[2]<=0 && 2< a.length<=100	a={2 145 681 341, 1 069 547 450, -2 147 482 624}			
			10	a.ref>0 && a[0]>0&& a[1]>0 && a[2]>0 && 2<a.length<=100&& a[0]<a[1]+a[2]&& [1]<a[0]+a[2] && a[2]<a[0]+a[1]	a={2,3,3}

5 结 论

本文分析了 JDart 在实际程序测试中所存在的问题,通过对 JDart 执行过程的分析,提出了一种对数组类型符号化进行优化的策略,并在 JDart 中实现。通过对比优化前、后程序测试的结果的比较,验证了其有效性。本文的优化策略可以有效提高 JDart 对涉及复杂对象程序测试的覆盖率。

但是,对于字符串、指针、对象等复杂对象的符号化仍存在问题,而且动态符号执行技术对比其他软件测试技术虽然有显著的优点,但是它仍旧面临着诸多的挑战,尚有很大的研究空间。

参考文献:

[1] Shamsoddin-Motlagh E. A Review of Automatic Test Cases Generation[J]. International Journal of Computer Applications, 2012, 57(13): 25-29

[2] Zheng W, Hierons R M, Li M, et al. Multi-Objective Optimisation for Regression Testing[J]. Information Sciences, 2016, 334/335: 1-16

[3] Anand S, Burke E K, Chen T Y, et al. An Orchestrated Survey of Methodologies for Automated Software Test Case Generation [J]. Journal of Systems & Software, 2013, 86(8):1978-2001

[4] Cadar C, Sen K. Symbolic Execution for Software Testing: Three Decades Later[J]. Communications of the ACM, 2013, 56

(2): 82-90

[5] Stephens N, Grosen J, Salls C, et al. Driller: Augmenting Fuzzing through Selective Symbolic Execution[C]//NDSS, 2016, 16: 1-16

[6] Yi Q, Yang Z, Guo S, et al. Postconditioned Symbolic Execution[C]//2015 IEEE 8th International Conference on Software Testing, Verification and Validation, 2015: 1-10

[7] Sobeih A, Marinov D. Optimized Execution of Deterministic Blocks in Java Pathfinder[C]//International Conference on Formal Methods and Software Engineering, 2006: 549-567

[8] Cadar C, Dunbar D, Engler D R. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs[C]//OSDI, 2008: 209-224

[9] Dimjašević M, Rakamarić Z. JPF-Doop: Combining Concolic and Random Testing for Java[C]//Java Pathfinder Workshop, 2013

[10] Luckow K, Dimjašević M, Giannakopoulou D, et al. JDart: A Dynamic Symbolic Analysis Framework[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2016: 442-459

[11] 周海将, 吴军华. 基于符号执行与混合约束求解的测试用例生成研究[J]. 计算机应用与软件, 2016, 33(6):23-26
Zhou Haijiang, Wu Junhua. On Test Case Generation Based on Symbolic Execution and Hybrid Constraint Solving. Computer Applications and Software,2016, 33(6):23-26 (in Chinese)

[12] 白晓颖, 黄军. 基于约束组合的测试用例生成[J]. 清华大学学报:自然科学版, 2017(3):225-233
Bai Xiaoying, Huang Jun. Case Generation by Constraints Combinatorial Testing[J]. Journal of Tsinghua University: Science and Technology, 2017(3):225-233 (in Chinese)

JDart-Based Test Cases Generation and Optimization

Wu Xiaoxue[1], Zheng Wei[2], Wang Peiyuan[2], Wang Peijia[2], Fan Songyu[2]

(1.School of Automation, Northwestern Polytechnical University, Xi'an 710072, China;
2.School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: Test cases play a crucial role in software testing, with the increasing complexity and scale of software, automatic test cases generation becomes increasingly important for software reliability and test efficiency. Symbolic-based test cases generation approach draws great attention due to its high reliability and there are already various kinds of tools introduced. However, most of these tools are C-oriented. JDart is a good open source Java-oriented symbol execution tool with excellent scalability. This paper aims to enhance the automatic test generation ability of JDart by designing and optimizing its array symbolization. The verification result shows that the optimization strategy proposed in this paper is effective in the test of the Jdart, it can effectively improve the coverage of Jdart on program involving complex object testing.

Keywords: test case generation; JDart, dynamic symbolic execution; optimization strategy, scalability, software reliability