计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(12):350−358 [doi: 10.15888/j.cnki.csa.008833]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于反射特征的 Android 测试用例自动生成①

闫怡梦, 赵瑞莲, 王微微, 尚　颖

(北京化工大学 信息科学与技术学院, 北京 100029)
通信作者: 赵瑞莲, E-mail: rlzhao@mail.buct.edu.cn

摘　要: 随着 Android 应用软件数量的急速增加, Android 应用质量的重要性越来越受到人们的重视. 测试是高质量软件的重要保证, 而测试用例生成技术是自动化测试的关键. 数据显示, 在 Google Play 中有将近 88% 的 Android 应用程序使用了反射. 然而, 现有的 Android 测试用例自动生成方法通常没有考虑反射技术的使用, 无法检测出反射隐藏的恶意行为. 为了进一步提高软件质量, 本文提出一种新的 Android 测试用例生成方法, 结合反射特征构造 Android 应用程序多粒度模型, 同时对反射关系进行分析, 生成能到达反射的调用路径, 并利用自适应遗传算法生成覆盖反射路径的测试用例, 对含反射特征的 Android 应用进行测试. 为验证本文方法, 分别从 Android 应用多粒度模型有效性及测试方法效率两方面对本文方法有效性进行评估. 实验结果表明, 本文提出的基于反射特征的 Android 测试用例自动生成方法对于反射的检测效果更好并且效率更高.

关键词: 反射机制; Android 应用; Android 多粒度模型; 自适应遗传算法; 测试用例生成

引用格式: 闫怡梦,赵瑞莲,王微微,尚颖.基于反射特征的 Android 测试用例自动生成.计算机系统应用,2022,31(12):350–358. http://www.c-s-a.org.cn/1003-3254/8833.html

Automatic Generation of Android Test Cases Based on Reflection Features

YAN Yi-Meng, ZHAO Rui-Lian, WANG Wei-Wei, SHANG Ying

(College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: With the rapid increase in the number of Android applications, more importance is attached to the quality of Android applications. Testing is an important guarantee for high-quality software, and test case generation technology is the key to automated testing. Data shows that nearly 88% of Android applications in Google Play use reflection. The existing automatic test case generation methods for Android, however, usually do not consider the use of reflection and cannot detect the malicious behavior hidden by reflection. To further improve software quality, this study proposes a new test case generation method for Android, which uses reflection features to construct a multi-grain model of Android applications. Meanwhile, it analyzes reflection relationships to generate call paths that can reach reflection and employs the adaptive genetic algorithm to generate test cases that cover reflection paths to test Android applications with reflection features. For verification, the effectiveness of this method is evaluated in terms of the effectiveness of the multi-grain model of Android applications and the efficiency of the test method. The experimental results reveal that the automatic test case generation method for Android, which is based on reflection features, is more effective and efficient in detecting reflection.

Key words: reflection mechanism; Android application; Android multi-grain model; adaptive genetic algorithm; test case generation

① 基金项目: 国家自然科学基金 (62077003)
收稿时间: 2022-03-25; 修改时间: 2022-04-22; 采用时间: 2022-04-29; csa 在线出版时间: 2022-07-29

1 引言

随着智能手机的普及, Android 应用的重要性不断增加. 目前, Google Play 应用商店的 Android 应用程序已超过 280 万个[1]. 尽管 Android 应用越来越受欢迎, 但其质量不容乐观. 对于 Android 应用程序自动化测试的需求也在不断增加[2]. 反射机制广泛地应用于 Android 应用程序中以增强 Android 应用程序功能. 但反射技术的使用可能会导致 Android 应用的各种安全隐患, 例如: 隐私泄露等. 因此, 研究面向反射的 Android 测试方法至关重要.

目前, 面向反射的 Android 测试技术已有初步进展, 其中, 基于模型的 Android 测试用例生成方法成为了主流. Android 应用中常用的模型主要包含 GUI 模型和方法调用图模型两种. GUI 模型是对 Android 应用图形用户界面的抽象[3], 例如: TrimDroid 利用有限状态机抽象表示 Android 应用的 GUI 及其行为[4]. 但 GUI 模型仅涉及前端界面表示, 难以反映 Android 应用代码中的反射特征. 方法调用图模型在 Android 中用于表示应用程序方法之间的调用关系. 由于 Android 事件驱动的特性, Android 应用的方法调用图模型存在多入口的特点. 例如: IntelliDroid[5] 通过读取 Android 组件并提取其生命周期方法, 同时搜索 Android 回调侦听器的实例化构建多入口的方法调用图. FlowDroid[6] 在构建方法调用图时采用相似的入口点发现机制. 不同的是, FlowDroid 通过构建虚拟的主方法生成单入口的方法调用图. DirectDroid[7] 采用与 IntelliDroid 相似的方法以生成多入口的方法调用图. 然而上述方法在构建方法调用图时并未考虑反射机制, 使得生成的方法调用图不完整. 此外, 反射所在方法内不同的执行路径均会导致调用结果的不同. 而方法调用图仅表征反射所在方法, 未深入分析反射方法的内部逻辑及执行流向, 不利于后续面向反射的 Android 测试生成.

面向反射的 Android 应用测试生成旨在生成可触发反射的测试用例. 其中, 基于符号执行的测试生成和基于搜索的测试生成方法的应用尤为广泛. 例如: IntelliDroid[5] 基于符号执行技术根据方法调用图生成到达反射的路径约束并利用约束求解器生成触发反射的输入. 然而, 由于输入类型支持有限、约束求解器的局限性等原因, IntelliDroid 生成的测试用例无法触发大多数的反射调用. 搜索算法因其简单高效等特点被广泛应用于 Android 测试用例生成研究中. 其中, 遗传算法因具备自寻优且全局搜索能力强的特点而尤为突出[8,9], 例如: Sapienz[10] 是一种基于多目标搜索的 Android 测试方法, 利用遗传算法自动探索和优化测试序列. 然而, Sapienz 在设计适应度函数时需要兼顾代码覆盖率, 测试序列长度及目标数量, 这 3 种因素的参数设置对于测试效果产生重要影响. EvoDroid[11] 利用遗传算法对方法调用图模型和接口模型进行演化搜索, 生成到达目标 API 的测试用例. 但由于在遗传算法中遗传操作算子的设计等问题, EvoDroid 生成测试用例的时间开销较大, 且测试效果不佳. 如何设计合适的适应度函数及遗传操作算子等以指导搜索过程是 Android 测试用例生成的关键.

综上所述, 现有的面向反射的 Android 测试技术存在如下问题: (1) Android 应用的模型未考虑反射机制导致生成的方法调用图不完整; 未深入分析方法内部逻辑及执行流向的问题; (2) Android 测试用例生成方法在适应度函数设计及遗传操作时存在的不足导致测试生成效率较低.

为此, 本文提出一种新的面向反射的 Android 测试用例自动生成方法, 基于反射特征并深入分析反射方法内部逻辑, 构建 Android 应用多粒度模型; 在此基础上, 并分析反射之间可能存在的关联关系, 指导生成到达反射的调用路径; 设计自适应遗传算法生成覆盖反射路径的测试数据, 触发反射行为, 为含反射的 Android 应用测试提供了一种有效的解决方案.

本文的主要贡献包括:

(1) 本文针对 Android 应用, 结合其方法调用图及方法内部逻辑, 构建面向反射的 Android 应用多粒度模型, 为 Android 测试生成奠定基础.

(2) 基于 Android 应用的多粒度模型分析反射之间可能存在的顺序或嵌套关系以指导反射路径的生成.

(3) 针对反射路径, 设计一种自适应遗传算法生成覆盖反射路径的测试数据. 本文在自适应遗传算法中设计多粒度适应度函数以指导搜索, 同时设计新的自适应交叉、变异算子, 以提高算法的搜索效率.

(4) 本文对 5 个 Android 应用程序进行实验分析, 发现本文方法构建的 Android 应用多粒度模型更能有效描述反射, 同时可以有效分析反射间可能存在的顺序或嵌套关系, 并且本文方法在测试用例生成过程中效率更高.

2 基于反射特征的 Android 测试用例自动生成方法

本节从方法框架、基于反射特征的 Android 应用多粒度模型构建、反射路径生成及基于自适应遗传算法的反射路径测试数据生成 4 个方面阐述本文方法.

2.1 方法框架

本文目标是研究一种基于反射特征的 Android 测试用例自动生成方法, 生成能够触发反射的测试用例, 并且该方法能够分析反射间可能存在的相互关系, 同时采用自适应的交叉、变异概率来提高自适应遗传算法的效率. 图 1 展示了本文基于反射特征的 Android 测试用例生成方法的框架结构, 由 3 个模块组成: ① 基于反射特征的 Android 应用多粒度模型的构建; ② 基于 Android 应用多粒度模型的反射路径生成; ③ 基于自适应遗传算法的反射路径测试数据生成. 下面将对这 3 个模块进行详细介绍.

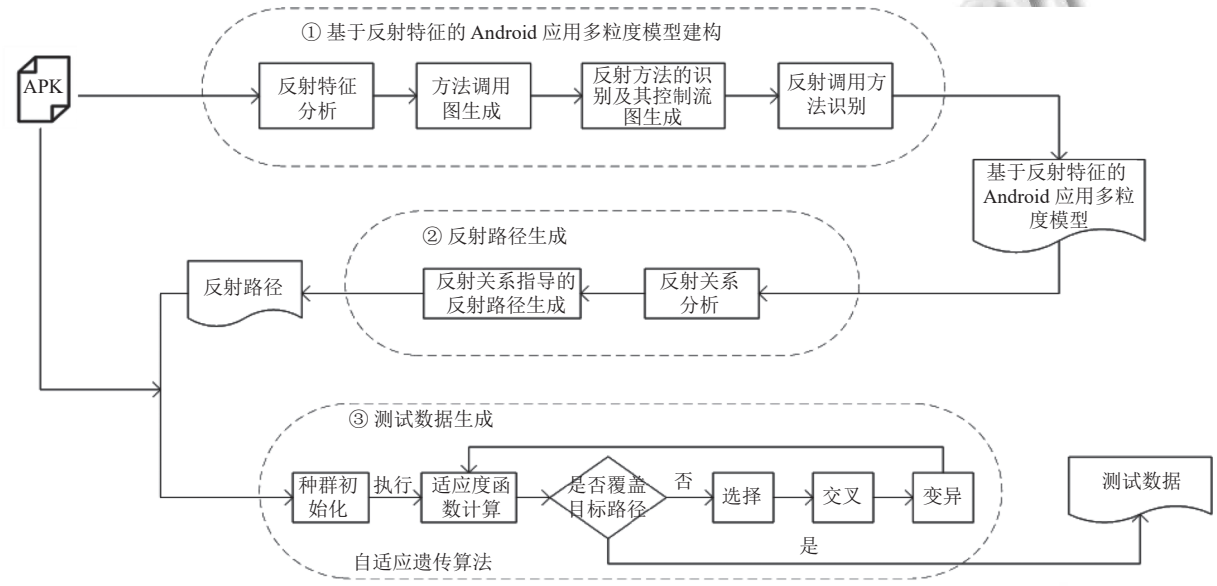


图 1 基于反射特征的 Android 测试用例自动生成方法框架

2.2 基于反射特征的 Android 应用多粒度模型构建

本节从反射特征基本概念及特征分析、Android 应用多粒度模型的定义及表示、基于反射特征的 Android 多粒度模型构建 3 个方面进行详细介绍.

2.2.1 反射机制的基本概念及其特征分析

反射机制允许 Android 应用程序在动态执行过程中加载类并获取类的详细信息, 从而操作类的对象进行方法调用. 通过 Java 反射机制, 可以在程序中访问已经装载到 JVM 中的 Java 对象, 实现访问、检测和修改描述 Java 对象本身信息的功能[12]. 根据图 2 代码示例可知, 应用程序在动态运行过程中利用反射技术加载类 (如: cName1), 创建类的对象并调用方法 (如: mName1).

通过对反射机制的分析, 可知道反射具有以下特征.

特征 1: 反射调用是一种间接调用

传统的 Java 调用通过对类进行实例化获得类的实例对象, 然后进行方法调用, 是一种直接调用方式 (显式调用). 然而, Java 反射机制是在动态执行过程中获取类的对象并进行方法的调用. 反射调用的 API 可以形式化地描述为: Method.invoke(obj, params), 其中 Method 对应实际的反射调用方法, obj 对应该反射调用方法所属类的实例对象. 因此, 反射调用是一种间接调用. 常用的 Android 测试工具是在构建方法调用图未考虑到反射机制的间接调用特征, 使得生成的方法调用图不够完整, 进而影响其检测的结果.

特征 2: 反射调用结果会受到其他包含反射调用目标参数的方法的影响

Android 应用程序中反射调用 API 可以形式化描述为 Method.invoke(obj, params). 不难发现, Method、obj 及 params 的不同均可能导致反射调用结果的不同. 然而 Method 对应的实际反射调用方法受到 getMethod 的影响, obj 对应的反射方法的类受到 forName 的影响. getMethod、forName 返回值的不同均可能导致反

射调用结果的不同. 因此, 忽视这些影响反射调用目标参数的方法很容易导致对于反射用结果的分析不够全面, 进而影响反射的检测效果.

```
1  Object createObj(String cName) {
2      Class c = Class.forName(cName);
3      return c.newInstance();
4  }
5
6  Method getMtd(String cName, String mName) {
7      Class c = Class.forName(cName);
8      return c.getMethod(mName, ...);
9  }
10
11 void foo(B b, C c, ...){
12     Object v = createObj(cName1);
13     if(...) {
14         A a = (A) v;
15         ...
16     }
17     else{
18         ...
19         Method m = getMtd(cName1, mName1);
20         m.invoke(v, new Object[]{b,c});
21     }
22 }
```

图 2 Android 应用程序代码示例

2.2.2 基于反射特征的 Android 应用多粒度模型的定义及表示

由反射特征 2 可知, 反射调用结果会受到反射调用目标参数的影响. 然而方法粒度的方法调用图模型无法有效表示方法内部各种可能影响反射调用结果的路径. 因此, 本文结合 Android 应用程序的方法调用图及反射所在方法 (简称: 反射方法) 的控制流图, 构建 Android 应用多粒度模型. 根据反射特征 1, 方法调用图模型在构建过程中未考虑到反射实际调用的方法. 因此, 本文通过静态字节码分析, 识别出实际的反射调用方法并添加相应的调用边, 得到基于反射特征的 Android 应用多粒度模型. 每个粒度的模型及最终的 Android 应用多粒度模型具体表示如下.

(1) Android 应用方法调用图及其表示

Android 应用的方法调用图反映了方法之间的调用及被调用关系. Android 应用多粒度模型的方法调用图模型表示为 $G1=<N1, E1, Mentry, Mexit>$. 其中, $N1$ 表示方法调用图的节点, $E1$ 表示方法调用图的边, *Mentry* 表示方法调用图的入口点, *Mexit* 表示方法调用图的出口点. $N1$ 对应的方法节点表示为 (*class_name*, *method_name*, *descriptor*, *is_reflection*). 其中, *class_name* 对应方法所属的类名, *method_name* 对应方法名, *descriptor* 表示方法描述符, *is_reflection* 用于表示该方法是否为反射方法. 本文假设调用方节点为 *src*, 被调用方节点为 *dst*, 则调用边 $E1$ 表示为<*src*, *dst*>.

(2) 反射方法的控制流图及其表示

控制流图是一个过程或程序的抽象表现. 控制流图的节点表示一个基本块, 边表示基本块的执行流向. Android 应用多粒度模型中的反射方法控制流图模型表示为 $G2=<N2, E2, Centry, Cexit>$. 其中, $N2$ 是控制流图节点的集合, $E2$ 是边的集合, *Centry* 表示反射方法的入口点, *Cexit* 表示反射方法的出口点. $N2$ 对应反射方法内的语句, $E2$ 表示为{<*src*, *dst*> | *src*, *dst*$\in N2$, 且 *src* 执行后可能立即执行 *dst*}.

(3)Android 应用多粒度模型的定义及表示

基于反射特征将方法调用图与反射方法控制流图结合后得到的 Android 应用多粒度模型定义为 $G=<N, E, Entry, Exit>$, N 表示 Android 应用多粒度模型的节点集, E 表示 Android 应用多粒度模型的边集, *Entry* 表示 Android 多粒度模型入口点集, *Exit* 表示 Android 多粒度模型出口点集. 其中, N 包含两种类型的节点, 一种是方法节点, 一种是控制流节点. E 对应的边集<*src*, *dst*>除了表示方法之间的调用关系, 以及控制流内部的执行流向, 还存在另外两种情况. 一种是 *src* 为方法节点, *dst* 为控制流图入口点, 对应反射方法及其控制流图入口点之间的调用边; 另一种是 *src* 为控制流图节点, *dst* 为方法调用图的方法节点, 对应反射点及其实际调用方法间的调用边.

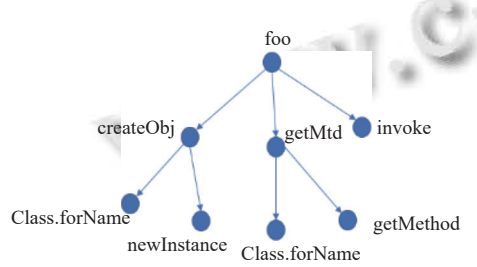
2.2.3 基于反射特征的 Android 应用多粒度模型构建

为构建基于反射特征的 Android 应用多粒度模型, 本文首先对 Android 应用程序进行预处理. 考虑到 Android 应用事件驱动的特性, 构建的 Android 应用程序的方法调用图存在多入口的特点[13]. 通过利用 Androguard[14] 为 Android 应用程序构建多入口的方法调用图, 然后进行反射方法识别, 对识别出的反射方法构建控制流图.

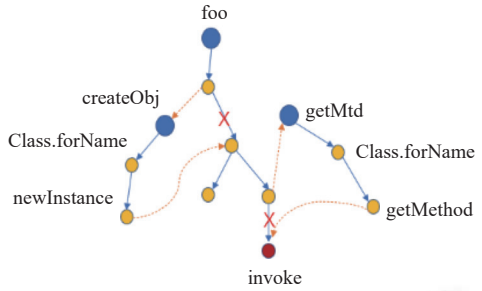
接下来, 本文对预处理阶段得到的方法调用图及反射方法控制流图进行多粒度模型融合以得到初始的 Android 应用多粒度模型. 本文对于反射方法及其对应的控制流图入口点之间添加调用边, 并在反射方法控制流图与方法调用图之间添加调用边及返回边以保持应用程序原有的逻辑关系. 根据反射特征 1 可知, 反射机制中实际的反射调用方法依赖于 forName 及 getMethod 的参数. 因此, 本文构造一个字节码解析器对反射 API 进行静态的字节码分析, 获取 forName 及 getMethod 参数, 确定其对应的方法 M, 然后在反射 API 与反射调

用方法 M 之间添加调用边, 进而得到最终的基于反射特征的 Android 应用多粒度模型.

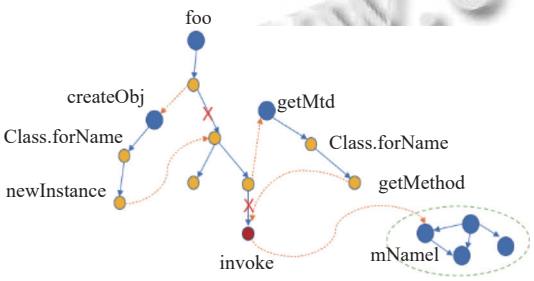
根据图 2 中所示代码, 本文对 Android 应用多粒度模型构建过程进行详细的描述. 如图 2 所示, 方法 foo 调用了方法 createObj 和 getMtd, 并根据 createObj 和 getMtd 的返回值进行了反射调用, 得到的方法调用图如图 3(a) 所示. 然后将方法调用图中的反射方法控制流图与方法调用图进行融合得到初始的 Android 应用多粒度模型, 如图 3(b) 所示. 通过静态分析可知, 反射 API 所对应的反射方法是动态加载类 cName1 中方法 mName1. 因此, 通过在 invoke 与 mName1 之间添加调用边得到最终的基于反射特征的 Android 应用多粒度模型, 如图 3(c) 所示.



(a) 方法调用图



(b) 初始的 Android 应用多粒度模型



(c) 基于反射特征的 Android 应用多粒度模型

图 3 基于反射特征的 Android 应用多粒度模型构建过程

2.3 基于 Android 应用多粒度模型的反射路径生成

考虑到不同的反射 API 之间可能存在一定的相互关系, 本文对反射间关系进行分析, 并根据关系分析结果指导反射路径的生成. 为便于描述, 本文进行如下定义.

定义 1. 反射路径. 从起始的方法节点到反射 API 之间的路径称之为反射路径. 反射路径同时包括方法粒度的方法调用序列及 CFG 粒度的控制流路径.

定义 2. 反射子路径. 从反射方法 CFG 起始节点到反射 API 之间的路径称之为反射子路径. 该路径只包括 CFG 粒度的控制流路径.

2.3.1 反射关系分析

本文通过字节码分析提取 forName 及 getMethod 等反射 API 的参数信息, 同时分析出每一个反射 API 的参数及返回值信息, 然后将获取的反射信息以五元组的形式存储 (S_m, R_m, T_m, *Parms*, *Rets*). 其中, S_m 是反射方法, R_m 是反射 API, T_m 是反射调用方法, *Parms* 对应反射 API 的参数信息; *Rets* 对应反射 API 的返回值信息. 反射方法 S_m 由反射方法所属的类名 *src_cls* 及反射方法名 *src_method* 组成, 反射调用方法 T_m 由反射调用方法所属的类名 *tar_cls* 及反射方法名 *tar_method* 组成. 本文对元组中所存储反射信息进行简单字符串匹配. 假设 Android 应用程序中存在反射点 *Ra* 与反射点 *Rb*, 如果存在 *Ra.tar_cls= Rb.src_cls&&Ra.tar_method=Rb.src_method*, 则 *Ra* 与 *Rb* 之间存在嵌套关系, 并且 *Ra* 与 *Rb* 之间存在一条 *Ra*→*Rb* 的调用路径; 如果 *Ra.Rets=Rb.Parms*|| *Ra.Rets=Rb.tar_cls*|| *Ra.Rets=Rb.tar_method*, 则 *Ra* 与 *Rb* 之间存在顺序关系, 并且 *Ra* 与 *Rb* 之间存在一条 *Ra*→*Rb* 的调用路径.

2.3.2 反射关系指导的反射路径生成

根据反射关系分析结果可知, 存在嵌套或顺序关系的反射 API 之间存在一条调用路径. 通过该路径, 可以从一个反射 API 到达另一个反射 API. 本文根据反射 API 之间是否存在相互关系定义以下不同的路径生成准则.

定义 3. 普通反射 API 的路径生成准则. 假设反射点 A 与其他反射点间不存在相互关系且路径起始点为 S_0, 则 A 的反射路径由 S_0 至 A 的方法调用序列 $L1$ 及反射子路径 $L2$ 组成, 即: LA: S_0→$L1$→$L2$→A.

定义 4. 基于顺序或嵌套关系的反射 API 的路径生成准则. 假设反射点 A 与反射点 B 之间存在顺序或嵌套关系且 A 的路径起始点为 S_0, 则 A 与 B 对应的反射路径由 S_0 至 A 的方法调用序列 $L1$、A 的反射子路径 $L2$、反射点 A 的调用方法至反射 B 的反射方法之间的方法调用序列 $L3$ 及反射点 B 的反射子路径 $L4$ 组

成, 即: L_{A+B}: S_0→$L1$→$L2$→A→$L3$→$L4$→B.

根据以上路径生成准则, 本文基于反射关系指导反射路径的生成.

对于与其他反射之间不存在相互关系的反射 API, 本文从该反射 API 开始进行反向控制流及数据流分析, 通过反向深度优先遍历, 得到所有从反射方法入口点到反射 API 的控制流路径, 然后进行数据流分析, 对不影响反射调用结果的路径予以删除, 以得到反射子路径. 由于反射路径由反射调用序列及控制流路径组成, 本文从反射方法节点开始进行反向深度优先搜索, 随机生成从 Android 应用程序入口点到反射方法的调用序列, 并将其与反射子路径结合, 以得到最终的反射路径.

对于存在顺序或嵌套关系的反射 API (假设反射点为 A 和 B), 本文分别从 A 与 B 所对应的 API 开始进行反向控制流及数据流分析, 生成到达反射点 A 与反射点 B 的反射子路径 $L2$、$L4$, 然后从 A 的反射方法开始进行反向深度优先搜索, 随机生成从 Android 应用程序入口点到反射方法的调用序列 $L1$, 接着从反射点 A 的调用方法开始, 进行深度优先搜索, 生成从反射点 A 的反射调用方法到 B 的反射方法间的调用序列 $L3$, 进而得到最终的反射路径, 即: L_{A+B}: $L1$→$L2$→$L3$→$L4$.

2.4 基于自适应遗传算法的反射路径测试数据生成

遗传算法是一种自适应寻优算法, 具有很好的全局搜索能力[15,16]. 相较于交叉、变异概率值固定的遗传算法, 概率自适应的遗传算法能够加快收敛速度, 提高搜索效率. 因此, 本文利用自适应的遗传算法生成覆盖反射路径的测试数据. 首先需要初始化种群. 种群中的个体由符号编码的染色体表示, 基因则由反射路径中的方法及其参数决定. 在得到初始化种群后, 执行被测程序, 对个体进行适应度评估, 判断其是否覆盖目标路径, 若覆盖目标路径, 则对该路径进行标记, 并继续分析目标路径集中未被标记的路径; 若当前种群未能覆盖目标路径, 则经过选择、交叉、变异等遗传操作实现种群的不断进化, 直到搜索到满足目标路径的最优解, 或者到达最大迭代次数.

2.4.1 基于反射路径的适应度函数设计

许多学者针对路径覆盖的测试用例生成研究提出了不同的适应度函数设计方法, 大致包括层接近度法和分支距离法[17]. 由于本文生成的反射路径由方法调用序列及控制流路径组成, 本文在设计适应度函数评估个体时, 相较于以往的适应度函数有所不同. 本文设计的适应度函数如式 (1) 所示:

$$F_{\text{global}} = mcg\_level + \left(1 - 1.01^{-f}\right) \tag{1}$$

其中, f 表达式如式 (2):

$$f = approach\_level + \left(1 - 1.01^{-branch\_distance}\right) \tag{2}$$

式 (1) 中, F_{global} 是计算得到的全局适应度函数值, mcg_level 表示在方法粒度当前路径与目标路径的接近程度. 更具体的, mcg_level 表示在方法粒度当前执行路径相较于目标路径未覆盖方法节点的个数. 在式 (2) 中, $approach\_level$ 表示在控制流粒度当前执行路径相较于目标路径未覆盖节点的个数. $branch\_distance$ 表示当执行路径与目标路径不同时, 第一个不同的分支谓词由假变真的距离.

综上可知, F_{global} 的值越小, 表示当前执行路径与目标路径越接近. 当 F_{global} 的值为 0 时, 说明目标路径被覆盖, 则返回其相应的测试数据, 并加入测试用例集保存.

2.4.2 算子的选择与设计

在利用自适应遗传算法进行测试数据生成的过程中, 本文采用排序选择法对种群中个体的适应度函数值进行排序, 同时采用交叉、变异算子的动态自适应方法提高算法的智能性. 本文采用较常用的单点交叉策略, 在染色体中随机选择一个交叉点进行交叉操作, 自适应交叉概率计算公式如式 (3):

$$Pci = (1 - normalize(Fi)) \times Favg + normalize(Fi) \tag{3}$$

其中, Pci 表示个体 i 的交叉概率, $normalize(Fi)$ 是第 i 个个体适应度函数值规范化后的结果, $Favg$ 是种群中个体适应度函数规范化后的平均值. 假设个体 i 与个体 j 之间进行交叉操作, 则$Pc = \max(Pci, Pcj)$.

通过基因变异产生新个体是保持物种多样性的重要手段. 本文设计了自适应变异概率值 Pm, 利用个体的适应度函数值动态调整变异概率. 自适应变异概率 Pm 计算公式如式 (4):

$$Pm = normalize(Fi) \times Favg^2 \times \rho \tag{4}$$

其中, ρ 是实数, 这里设置为 0.5. 面向反射路径的测试数据生成算法如算法 1 所示.

算法 1. 面向反射路径的测试数据生成算法

输入: 待测 Android 应用, 基于反射特征的 Android 应用多粒度模型, 一组反射路径 Target_Paths

输出: 一组覆盖 Target_Paths 的测试数据

```
声明: Target: 一条反射路径
MaxAttempts(): 试图覆盖 Target 却超过了最大迭代次数时, 返回 True 的函数
OutOfTime(): 当超过时间限制时, 返回 True 的函数
1 while(some(Target, unmarked)∈Target_Paths) and not OutOfTime():
2    从 Target_Paths 中选择未被标记的 Target
3    pop = Init_Population(Target, N) //生成大小为 N 的初始种群
4    F_global = Global_fitness_Evaluation(pop)
       // 计算当前种群的适应度函数值
5    while(Target not marked and not MaxAttempts()):
6      if(Target not covered):
7        根据适应度函数值排序
8        Indi, Indj=Select_Operator (pop, F_global)
9        Indi, Indj=Crossover_Operator(pc, Indi, Indj)
10       Indi, Indj =Mutation_Operator(pm, Indi, Indj)
11       NewInds ← Indi, Indj
12        Executor(NewInds)
13       F_global = Global_fitness_Evaluation(NewInds)
14       iteration++
15     else:
16       标记 Target 并返回覆盖 Target 的测试数据
17    end while
18   返回 Target_Paths 对应的测试数据
19 end while
```

3 实验分析

3.1 实验设计

(1) 研究问题

为验证本文所提出的方法的有效性, 实验部分主要解决以下 3 个问题.

1) 本文构建的基于反射特征的 Android 应用多粒度模型是否有效描述反射?

2) 本文对于反射关系的分析是否有效指导反射路径生成?

3) 本文设计的多粒度适应度函数及自适应的交叉、变异算子是否有效指导反射路径的测试数据生成?

(2) 实验对象

本文在 GitHub 挑选 5 个 APK 作为实验对象. ContactReader3、InvokeDeviceID、InvokeDeviceID2、OnlySMS、SharedPreference 均由 Java 语言编写实现. 为分析反射及反射相关信息, 本文选取的待测 APK 均包含反射. 表 1 中提供了这 5 个 Android 应用程序的反射相关信息, 包括反射路径数、关系分析后的反射路径数及顺序和嵌套关系分别对应的反射路径数.

(3) 评估指标

1) 本文将反射路径数 (ref_paths) 作为度量指标以评估基于反射特征的 Android 应用多粒度模型的有效性及反射关系分析方法的有效性.

以往的 Android 测试用例生成方法在处理反射时把反射 API 当作普通的目标 API 进行测试用例的自动生成. 然而, 本文根据反射特征分析结果了解到, 传统的 Android 测试用例生成方法在分析反射路径时, 并没有分析实际的反射调用方法. 此外, 应用程序利用反射机制加载不同的类、调用不同的方法都会导致反射调用结果的不同. 因此, 仅仅生成触发反射 API 的测试用例是不够的, 还需要分析反射间关系及所有可能的反射路径. 为验证 Android 应用多粒度模型的有效性及反射关系分析方法的有效性, 本文将反射路径数 (ref_paths) 作为度量指标. 更详细地, 可以划分为 Android 应用程序所对应的反射路径总数 ref_paths、关系分析后的反射路径总数 ref_paths_after、顺序关系对应的反射路径数 ref_paths_order、嵌套关系对应的反射路径数 ref_paths_nested 及不存在相互关系的普通反射 API 所对应的反射路径数 ref_paths_normal.

2) 本文将生成覆盖 Android 应用程序反射路径的测试数据所消耗的时间 Time 作为度量指标以评估本文提出的自适应遗传算法的有效性.

本文将基于自适应遗传算法的 Android 测试用例生成方法与基于传统遗传算法 (交叉、变异概率值固定的遗传算法) 的 Android 测试用例生成方法所消耗的总时间进行对比, 进而分析本文概率自适应的遗传算法的有效性.

3.2 实验结果与分析

对于研究问题①, 本文对 5 个 Android 应用程序进行分析, 获得其相应的反射路径数, 如表 1 所示, 然后将获得的反射路径数与 StaDynA 方法[18] 得到的反射路径数进行对比, 发现本文方法在关系分析前获取的反射路径总数相比于 StaDynA 有所提升, 因此本文构建的基于反射特征 Android 应用多粒度模型是有效的.

对于研究问题②, 本文对 5 个 Android 应用程序的反射间关系进行了分析, 不仅得到关系分析后的反射路径总数, 还得到顺序或嵌套关系的反射所对应的反射路径数. 根据表 1 所示, 关系分析后的反射路径数相比关系分析前的路径数有所减少, 同时本文能够获得每种关系所对应的反射路径数量. 其中, OnlySMS 和 SharedPreference 存在顺序关系及嵌套关系的反射

路径数均为 0. 本文对这两个 Android 应用程序进行手动分析及验证, 发现 OnlySMS 及 SharedPreference 中的确不存在顺序关系或嵌套关系的反射. 因此, 本文方法可以有效检测到反射之间存在的相互关系, 从而说明本文所提出的反射关系分析方法是有效的.

对于研究问题③, 本文将基于传统遗传算法 (交叉概率值为 0.75, 变异概率值为 0.08) 的 Android 测试用例生成方法总的消耗时间 Time 与本文提出的基于概率自适应的遗传算法的测试用例生成方法所消耗的总时间 Time 进行对比, 如表 2 所示. 分别针对 5 个 Android 应用程序生成覆盖全部反射路径的测试数据, 并将每个 Android 应用程序所消耗的总时间与基于传统遗传算法的 Android 测试用例生成方法进行对比分析, 发现本文所提出的基于自适应遗传算法的反射路径测试数据生成方法所消耗的时间小于基于传统遗传算法的反射路径测试数据生成方法所消耗的总时间. 因此, 本文采用的基于自适应遗传算法的测试用例自动生成方法能够有效提高算法的搜索效率, 进而说明本文基于自适应遗传算法的 Android 测试用例生成方法能够有效指导反射路径的测试数据生成.

表 1 本文方法与 StaDynA 方法关于反射路径数的对比

APPs	StaDynA	本文方法				
	ref_paths	ref_paths	ref_paths_after	ref_paths_order	ref_paths_nested	ref_paths_normal
ContactReader3	2	2	1	1	0	0
InvokeDeviceID	1	1	1	1	0	0
InvokeDeviceID2	2	2	1	0	1	0
OnlySMS	1	2	2	0	0	2
SharedPreference	2	4	4	0	0	4

表 2 本文方法与基于传统遗传算法的 Android 测试用例生成方法的时间对比 (s)

方法	ContactReader3	InvokeDeviceID	InvokeDeviceID2	OnlySMS	SharedPreference
传统遗传算法	11.244	19.652	28.263	26.733	32.494
本文方法	9.326	16.347	23.751	22.596	26.182

4 结论与展望

本文提出一种基于反射特征的 Android 测试用例自动生成方法. 该方法对反射特征进行分析, 利用反射方法的控制流图对 Android 应用程序的方法调用图进行补充, 并识别实际的反射调用方法, 添加相应的调用边, 最终得到基于反射特征的 Android 应用多粒度模型. 该方法对反射间可能存在的相互关系进行分析, 并根据关系分析结果指导反射路径的生成, 最后利用自适应遗传算法生成覆盖反射路径的测试数据. 在对反射进行测试用例自动生成的过程中, 本文构建的基于反射特征的 Android 应用多粒度模型相较于传统的方法调用图模型更加完善, 解决了以往方法在处理反射问题时方法调用图不完整的问题, 进而得到更加全面的反射路径. 此外, 对于反射间关系的分析使得本文方法能够生成同时触发多个反射的测试用例. 本文在利用自适应遗传算法生成测试数据时, 不仅设计多粒度的适应度函数以指导搜索, 而且设计新的自适应交叉、变异算子, 提高了算法的搜索效率.

下一步工作将结合本文基于反射特征构建的 Android 应用多粒度模型对反射引起的各种恶意行为进行分析, 进而开发出针对 Android 应用程序反射代码的恶意行为检测工具.

参考文献

1 Moreno IA. Search-based test generation for Android apps. 2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). Seoul: IEEE, 2020. 230–233.

2 Cai TQ, Zhang Z, Yang P. Fastbot: A multi-agent model-based test generation system Beijing Bytedance Network Technology Co. Ltd. Proceedings of the IEEE/ACM 1st International Conference on Automation of Software Test. Seoul: ACM, 2020. 93–96.

3 Su T, Meng GZ, Chen YT, *et al*. Guided, stochastic model-based GUI testing of Android apps. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 245–256.

4 Mirzaei N, Garcia J, Bagheri H, *et al*. Reducing combinatorics in GUI testing of Android applications. Proceedings of the 38th International Conference on

Software Engineering (ICSE). Austin: ACM, 2016. 559–570.
5 Wong MY, Lie D. IntelliDroid: A targeted input generator for the dynamic analysis of Android malware. Proceedings of the 23rd Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2016. 21–24.
6 Arzt S, Rasthofer S, Fritz C, *et al*. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. ACM SIGPLAN Notices, 2014, 49(6): 259–269. [doi: 10.1145/2666356.2594299]
7 Wang XL, Yang YX, Zhu SC. Automated hybrid analysis of Android malware through augmenting fuzzing with forced execution. IEEE Transactions on Mobile Computing, 2019, 18(12): 2768–2782. [doi: 10.1109/TMC.2018.2886881]
8 吴昊, 李浩然, 万交龙. 对于测试用例生成的遗传算法改进. 计算机系统应用, 2016, 25(8): 200–205. [doi: 10.15888/j.cnki.csa.005307]
9 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. 计算机系统应用, 2013, 22(7): 90–94. [doi: 10.3969/j.issn.1003-3254.2013.07.020]
10 Mao K, Harman M, Jia Y. Sapienz: Multi-objective automated testing for Android applications. Proceedings of the 25th International Symposium on Software Testing and Analysis. Online: ACM, 2016. 94–105.
11 Mahmood R, Mirzaei N, Malek S. EvoDroid: Segmented evolutionary testing of Android apps. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong: ACM, 2014. 599–609.
12 乐洪舟, 张玉清, 王文杰, 等. Android 动态加载与反射机制的静态污点分析研究. 计算机研究与发展, 2017, 54(2): 313–327. [doi: 10.7544/issn1000-1239.2017.20150928]
13 Yang SQ, Yan DC, Wu HW, *et al*. Static control-flow analysis of user-driven callbacks in Android applications. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Florence: IEEE, 2015. 89–99. [doi: 10.1109/ICSE.2015.31]
14 AndroGuard: Reverse engineering, malware and goodware analysis of Android applications. https://code.google.com/p/androguard/.
15 王佳楠, 王玉莹, 何淑林, 等. 基于改进遗传算法优化 BP 神经网络的土壤湿度预测模型. 计算机系统应用, 2022, 31(2): 273–278. [doi: 10.15888/j.cnki.csa.008324]
16 胡程鹏, 薛涛. 基于遗传算法的 Kubernetes 资源调度算法. 计算机系统应用, 2021, 30(9): 152–160. [doi: 10.15888/j.cnki.csa.008062]
17 包晓安, 熊子健, 张唯, 等. 一种基于改进遗传算法的路径测试用例生成方法. 计算机科学, 2018, 45(8): 174–178, 190.
18 Zhauniarovich Y, Ahmad M, Gadyatskaya O, *et al*. StaDynA: Addressing the problem of dynamic code updates in the security analysis of Android applications. Proceedings of the 5th ACM Conference on Data and Application Security and Privacy. San Antonio: ACM, 2015. 37–48.

(校对责编: 牛欣悦)