

**第二十八届“冯如杯”学生学术科技作品竞赛项目论文**

**Android 自动化调试技术开发**

摘要

随着近年来移动端的蓬勃发展，安卓应用开发已成为计算机科学相关的重要领域，对其调试水平的水准要求也随之水涨船高。目前行业内流行着的各类调试工具，如Monkey，Monkey Runner 等虽特点不同功能各异，但都有其弊端和局限性。如Monkey类压力测试工具返回信息冗长，可直接使用性差，Appium类脚本运行工具覆盖性差，需要人工预置测试代码等。在此大环境下，本小组基于指导教师的先前成果改良并开发了一款依赖于Monkey的自动化调试工具，本工具借鉴了国内外先进的Monkey返回信息精简思路，采用了兼具效率和精度的LHDD算法作为错误定位系统的内核，使得其较传统DD算法在各方面性能上都有所提升；同时，本工具还创新性的集成了开源录像重放工具Reran，使得工具在问题序列精简的过程中有更优的可视化效果，并且获得更为合适细度的目标问题序列。

**关键字：**安卓开发，自动化调试，重放工具，LHDD算法

Abstract

In recent years, the technology of smartphone is rising and Android application development is becoming a vital part of portable computer science. For more and more apps approaching market, debugging technology also requires to reach a higher level. Though there are already several debugging tools for Android developers now, for example Monkey , Monkey Runner , Appium etc. But each of them features specific drawbacks and limitations . Monkey , as a stress testing tool , returns redundant operating list which is confusing for developers and need further manufacturing. Monkey Runner , a script following tool , is devasting in dealing with covering works. Against this backdrop, we develop an Android debugging tool which base on Monkey . During developing ,we adopt innovative sequence reducing algorithm: LHDD，which performs better than tradition reducing algorithm DD in both efficiency and accuracy. Further more , we integrate an open replay tool RERAN in our project in order to optimizing it’s visualization performance and getting a more suitable target order sequence.

**Keywords**

**Android development , Automatic debugging tool , Replay tool, LHDD algorithm**

目录

摘要i

Abstractii

引言1

第一章 工具核心算法介绍2

1.1 GUI状态层次树2

1.2 经典DD的精简过程3

1.3 LHDD的精简过程3

第二章 有效事件序列简化工具的框架4

2.1 建立GUI状态层次树4

2.2 Local Hierarchical Delta Debugging Algorithm (LHDD) 5

第三章 有效事件序列简化工具的框架7

结论9

参考文献10

引言

在近几年，移动互联网业飞速发展。移动设备应用的数量和复杂性都快速增长。在Google Play中有超过2,600,000的Android应用。

为了让终端用户满意，移动应用的开发者必须要提升应用的质量。移动测试是达到这个目标的一个重要方法。多种测试方法被提出，包括压力测试, GUI traversal-based testing,和search-based testing。压力测试代表性的工具有Monkey类工具，包括Android系统中内置的Monkey工具和第三方改进的版本。改进的Monkey工具由于其简单性、有效性和广泛的适用性，被广泛应用于基于云的移动测试平台。

这些压力测试方法的主要限制是他们经常在触发failure之前产生大量的输入事件，这样会导致接下来的调试任务难以进行。这时，我们期望简化输入序列并产生相同的failure。

Delta debugging (DD)被用来精简传统的应用，网络应用和编译器的测试输入。也有一些研究基于DD思想减少Android应用程序输入事件序列的方法。然而，在一般情况下，DD方法会很慢产生精简的输入序列。DD策略的问题是它的划分策略没有意识到输入事件与终端用户的交互，这可能导致在重放过程中产生大量失败的试验（即不能触发failure）。

Hierarchical Delta Debugging揭示了测试用例的层次结构并根据这种结构应用DD来提高DD的效率。我们观察到Android输入事件序列可以基于交互会话和子会话分层构造。

我们对事件序列简化问题的理解是，长输入事件序列通常包含着与终端用户的小交互会话子序列。这些事件经常可以以很高的概率减少。我们进一步观察到，输入事件的关系反映在它们对应的GUI状态的层次关系中。因此，我们试图通过建立和分析GUI状态层次树来寻找输入事件之间的层次关系。

基于输入序列的表示意义，我们提出了Local Hierarchical Delta Debugging (LHDD)算法来利用输入序列代表的结构，以提升简化效率并且不显著的降低简化的效果。

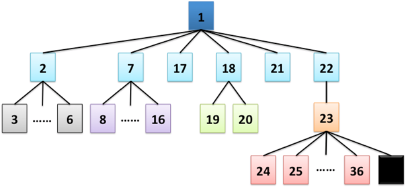
本工具采用了近些年提出的一种新的GUI状态层次树模型来模拟Android输入事件，并且提出了一个有效的HDD算法来简化Android输入事件跟踪。同时我

们对够早的一些样例进行了实验，评估了工具的有效性和改进效率。

一、工具核心算法介绍

1.1 GUI状态层次树

为了理解GUI输入事件的层次结构，我们必须建立一个局部GUI状态层次树，并将输入事件映射到它。这里的“部分”表示GUI层次结构树只反映与当前输入事件序列相关的GUI层次结构的那些部分，以简化操作。图1显示的GUI状态层次树构建了Android应用dalvikexplorer实际崩溃的序列。



**图** 1 **GUI状态树图**

在建立GUI状态层次树时，我们使用GUI状态的层次关系确定它们相应事件的层次关系。在这种状态层次树中，每个i 节点不仅表示崩溃序列中的事件i，而且在到达事件i之前的GUI状态。在GUI状态层次树中的边表示两个对应GUI状态之间的父子关系（层次关系）。

如果一个新的节点n与从根节点到其上一个节点的路径上的任何节点的状态不同，那么将节点n定义为其上一个的节点的子节点；否则，n被定义为具有相同状态的等效节点的新（右）兄弟节点。在图1，状态层次树中我们用相同的颜色表示的等价状态（兄弟节点），最后一个没有任何序号的节点来代表状态“crash”。在构建GUI层次树的过程中，新的事件会在右子树的生长。因此，崩溃节点（即事件序列的最后事件）总是在整个树的右子树的最底层最右边的节点。此外，崩溃节点的父节点和祖先节点始终是相应层的最后节点。

我们已经构建了一个增强的Monkey工具来在测试期间记录应用程序GUI状态。GUI状态的定义可以有不同的粒度。在目前的版本中，我们使用Activity ID代表一个GUI状态，这是一个轻量级的解决方案。使用更精细的GUI状态（例如GUI结构）定义可能会导致不同的还原结果，这方面的权衡是值得探索的。

因此，我们的崩溃序列不仅包含输入事件序列，还包含相应的GUI状态。更具体地说，我们的崩溃事件序列 ，和它对应的GUI状态序列 。在接收事件时，状态从转换到。

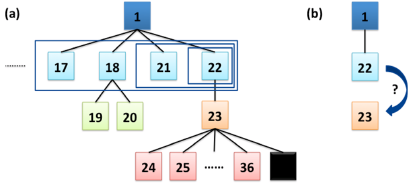
GUI状态层次树的构造如下：对于具有相同编号i的每个状态和事件构造一个新的节点i。然后节点n与从根到其之前节点的每个节点（状态）相比，如果等于任何节点m，则将n节点添加为m的兄弟节点，否则i添加为的子节点的。

1.2 经典DD的简化过程

Delta Debugging（DD）算法不使用GUI状态的层次关系[ 27 ]。将一个输入事件序列分成大小相等的子序列，并逐渐减少粒度。在我们的实验中对于长度为36的示例崩溃序列，DD算法执行了38次试验，最终获得简化测试输入序列，总共花费3分钟15秒。我们注意到，这38次试验中的许多都产生了无效的序列，降低了效率。

1.3 LHDD的简化过程

正如1.1部分所讨论的，崩溃节点的父节点和祖先始终是它们相应层的最后一个节点（例如，节点22和23）。我们可以从示例的GUI层次结构树中看到，崩溃节点的这些祖先节点（例如，节点22和23）是关键的，因为在这些节点上，被测试的应用程序过渡到接近崩溃节点的下一级GUI状态。基于这一观察，该LHDD算法采用启发式搜索：还原事件的顺序时保存一层的最后一个节点到其下一层的第一个节点的转换。例如，在图1的GUI树中，我们只需要检查第一层的转换是否成功（蓝色）（图2（b））和第二层的转换是否成功（橙色）。在最后一层层，LHDD检查崩溃是否发生。



**图 2** **LHDD算法示意**

通过这种方法，我们将简化过程从全局优化问题有效地转化为局部优化问题。简化过程不需要检查最后一个节点的子树，但是除了最后一个节点以外的节点的子树应该被检查。例如，简化第一层（蓝点）时，LHDD不对根的子树节点22检查，但对节点2, 17和18的子树的检查。根据我们的经验，最后一个节点的子树通常规模很大。在这样的情况下，LHDD直观的节省了大量的时间。

为了进一步优化局部简化的过程，在每一层LHDD采用启发式，然后执行DD简化。由于当前层上的简化事件序列必须包括最后一个节点，所以它增加2的幂次来选择从最后一个节点到前面节点（事件），不断尝试执行，在发现第一个子序列仍能到达下一层的第一个节点事件时停止。如图4（a）所示，预选过程将依次尝试事件序列, 和，直到事件序列被局部简化标准接受为止。最后，LHDD仅用了16秒进行4次试验得到与DD相同的简化结果，这比以前的DD算法更有效。

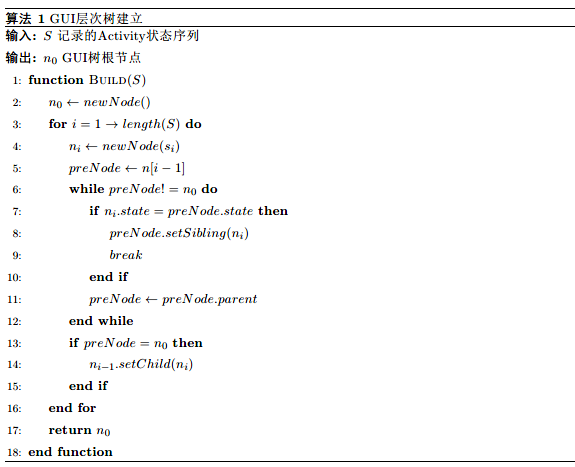
二、有效事件序列简化工具的框架

2.1建立GUI状态层次树

该算法以输入事件和状态序列来构建GUI状态层次树。正如之前所讨论的。状态的崩溃序列是，代表状态，对应着事件序列，其中每个事件触发从状态到的转换。该算法为每个状态（第2行到第4行）构建一个新节点。然后它与其之前前的节点

到根节点相比（第5行到第12行）。如果等于任何节点，则将它添加为

该节点的兄弟节点。否则，它将作为之前一个节点的子节点（第13行和第15行）。



**图 3** **GUI树建立算法伪代码**

2.2 Local Hierarchical Delta Debugging Algorithm (LHDD)

LHDD算法是基于HDD的，并且加入了两处优化。第一个优化是添加预选择（第7行到第23行）的过程，在此过程中不断添加的节点，直到找到触发下一级层活动的事件为止。第二优化是checkEvents()检查能否从本层最后一个节点到下一层第一个节点，这将保存包含最后一个节点的子树。inlevel\_hdd()是一种类似DD算法中不断减小序列粒度的方法。最后，checkevents()对于局部检查，该方法只检查是否成功转换到下一个活动，对于最后一层的检查，该方法检查是否触发了崩溃。

给出LHDD算法伪代码如下：



**图 4 LHDD算法伪代码**

三、实验和结果分析

我们使用MI5作为实验机，手机系统为Android 6.0.1。我们将模拟两次事件的时间间隔设置为800ms以确保有充足的时间使事件进行。测试的应用全部为真实的手机应用。

详细数据如下：

**表 1** **DD算法实验结果**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据序号 | 1 | 2 | 3 | 4 | 5 |
| 原始长度 | 131 | 77 | 205 | 352 | 461 |
| DD后长度 | 5 | 5 | 4 | 4 | **4** |
| 时间/ms | 905755 | 380961 | 1271780 | 9463640 | 1164383 |
| 数据序号 | 6 | 7 | 8 | 9 | 10 |
| 原始长度 | 49 | 216 | 678 | 148 | 230 |
| DD后长度 | 4 | 5 | 5 | 4 | 5 |
| 时间/ms | 288420 | 1005728 | 1736551 | 381406 | 958302 |

**表 2** **LHDD算法实验结果**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据序号 | 1 | 2 | 3 | 4 | 5 |
| 原始长度 | 131 | 77 | 205 | 352 | 461 |
| LHDD后长度 | 5 | 5 | 4 | 5 | 4 |
| 时间/ms | 557453 | 136246 | 146356 | 452158 | 281420 |
| 数据序号 | 6 | 7 | 8 | 9 | 10 |
| 原始长度 | 49 | 216 | 678 | 148 | 230 |
| LHDD后长度 | 4 | 5 | 6 | 4 | 5 |
| 时间/ms | 125949 | 871489 | 422621 | 57998 | 169772 |

**图 5 DD与LHDD算法效率对比**

从数据表格与柱状图中我们可以看出，DD与LHDD算法的确能够大幅度减小产生crash的事件序列，且我们对DD算法加入局部搜索后的LHDD算法精简的序列长度基本与DD算法相同，而LHDD算法需要的时间却远远小于DD算法。在第4组数据中我们可以看到，LHDD有较令人满意的稳定性，遇到极端情况精简的速度也不会很慢。

结论

本小组开发的安卓自动化开发工具可以较好调用Monkey并精简其返回事件序列以完成对产生Crash区段的辨析定位。同时，实验也表明我们所采用的LHDD精简算法内核于精度和效率上都优于传统的DD算法；录像工具RERAN与程序代码主体的集成度良好，既可以重放在操作中用户手动产生的问题，也可以对压力测试返回序列和经过切割后的序列进行浮现。整体而言，此自动化调试工具在工作各维度都表现良好，虽然由于其主要目标人群是有经验的调试者，导致操作较为复杂并要进行预先配置，但就综合性能上与现存各个人级免费调试工具相比都有其独特的竞争力。

参考文献

[1] T. Azim, I. Neamtiu, Targeted and Depth-first Exploration for Systematic Testing of Android Apps. In Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA2013), New York, NY, USA: ACM, pp. 641–660, 2013.

[2] L. Clapp, O. Bastani, S. Anand, A. Aiken. Minimizing GUI event traces. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016). ACM, New York, NY, USA, 422-434. 2016.

[3] A. Machiry, R. Tahiliani, M. Naik, Dynodroid: An Input Generation System for Android Apps. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013), New York, NY, USA: ACM, pp. 224–234, 2013.

[4] G. Misherghi, Z. Su. HDD: Hierarchical Delta Debugging. In Proceedings of the 28th International Conference on Software Engineering. Shanghai, China: ACM, pp. 142-151, 2006.

[5] A. Zeller, R. Hildebrandt. Simplifying and Isolating Failure-Inducing Input. IEEE Transactions on Software Engineering, 28(2): 183-200, 2002.

[6] Mobile Testing Center of Baidu. http://mtc.baidu.com, last access, 2017.

[7] UTEST platform of Tencent. http://utest.qq.com, last access, 2017.