规则驱动的 Android 应用 DFS 测试技术*

叶佳[1], 葛红军[2+], 曹春[2], 朱晋[1], 张营[2]

1. 中兴通讯股份有限公司 终端事业部, 江苏省南京市 210012

2. 南京大学 计算机科学与技术系, 江苏省南京市 210046

Rule-driven DFS Testing for Android Application*

YE Jia[1], GE Hong-Jun[2+], CAO Chun[2], ZHU Jin[1], ZHANG Ying[2]

1. Test Dept.Ⅲ Terminal Business Division, ZTE Corporation, Nanjing 210046

2. Department of Computer Science and Technology, Nanjing University, Nanjing 210046

+ Corresponding author: Phn +86-18-0520-95653 , E-mail: ghj_nju@163.com, http:// cs.nju.edu.cn/

Abstract: Automated GUI testing is important to Android application testing. Several technologies for automated Android GUI testing have been studied. Testing technology which is based on DFS exploration has been extensively used among them. However, the existing DFS testing technology is still inefficient and has low testing coverage. We propose an improved approach by driving the DFS automated exploration with external predefined rules to increase the efficiency and coverage. We implemented a testing tool called RDTA based on our approach and evaluated the performance of RDTA comparing to Monkey and original DFS without rules. The result verified the effectiveness of our approach.

Key words: Android testing; DFS; rule-driven; efficiency; coverage

摘 要: GUI 自动化测试是 Android 应用研究领域的重要组成部分，针对 Android 应用的 GUI 测试技术得到了广泛的研究。其中，基于 DFS 算法的 GUI 遍历测试技术得到了广泛的应用。然而，现有的 DFS 测试技术却仍然具有效率低下、覆盖率较低的问题。我们提出了结合外部预定义规则来驱动 DFS 自动化遍历的改进方法，来提高 DFS 自动化遍历的效率和覆盖率；基于规则驱动的改进方法实现了 RDTA 测试工具，进行了与 Monkey 以及无规则驱动下的 DFS 的对比实验，验证了该方法的有效性。

关键词: Android 测试; DFS; 规则驱动; 测试效率; 覆盖率

1 引言

近年来，智能手机在人们的日常生活中日益普及。据 IDC 的调查，Android 设备占据了约 85%的市场份额，是目前主流的智能手机平台。然而，由于 Android 版本和设备型号的多样性，Android 应用快速的更新迭代周期以及手工测试较低的效率，很多应用往往没有被充分测试就发布出来，这导致其

中很可能隐藏着错误(Bug)。

近年来，为提高 Android 应用测试的效率和应用软件的可信水平[15]，自动化测试技术得到了广泛的研究。根据自动化测试技术遍历应用 GUI 的不同策略[7]，可以将测试工具分为以下几类：

1、随机探索策略：测试工具如 Monkey 和 Dynodroid[12]，它们在测试过程选择的探索策略是随机算法，它们的目的往往不是系统化地对应用进行测试，而是通过产生大量的随机输入序列，对应用进行压力测试。

2、基于模型的探索策略：测试工具如 GUIRipper[2]、A3E-Depth-First[4]等，它们在自动化遍历测试过程中对应用的 GUI 状态和状态的转移进行建模[9]，最终生成测试过程的状态转移有向图或者有限状态机，并且记录测试的路径作为脚本。

3、系统的探索策略：测试工具如 Evodroid[6]和 ACTEve，这一类测试工具采用符号执行[5]、演化算法等方式来引导探索的路径，它们能比随机测试探索到更深入的路径分支。

基于模型的自动化测试算法应用较为广泛，在[1][3][4][10]中都是使用了 DFS 算法作为测试策略，这些论文中的工作随着应用的复杂性越来越高，测试过程中遇到状态爆炸的问题，导致测试过程中存在着效率较低的问题；在[8]提出了自动化的测试过程应该在用户指导下提高测试效果。

在测试实践中，测试人员往往还通过编写测试脚本的方式来进行 Android 应用测试[13]，测试工具包括 Android 原生的 Instrumentation 框架以及 MonkeyRunner 工具。编写脚本的方式能够很好地利用测试人员的对应用的先验知识，约束测试的路径，达到定向测试的效果；但是编写脚本的方式测试人员的工作量较大，而且随着界面的变化，脚本的复用率低，需要重新编写新的脚本。

本文中，我们提出了一种结合 DFS 自动化测试与规则的测试方式。本文将详细讨论如何通过定义外部规则的方式用测试人员的先验知识来驱动和约束测试过程，提高测试的效率和覆盖率。

2 Android 应用的 DFS 测试

2.1 相关概念

活动（Activity）：在一个 Android 应用中，Activity 是最重要的组件。Activity 为应用提供了一个屏幕，它其中包含的一些界面组件可以监听并且处理用户的事件做出响应。在 Android 测试中，Activity 的覆盖率通常衡量着对整个应用所有的测试的覆盖率。

界面组件（Widget）：每一个界面组件是组成 Android 的一个界面元素，都是可视化可用来与用户交互的组件，例如 ImageView、TextView、ListView、LinearLayout 等。界面组件之间构成了一个分层的树形结构。

驱动事件（Event）：不同的界面组件上可以根据用户的操作(Action)产生不同的响应，在界面组件上可以进行的例如点击、长按、滑动、文本输入等都是 Android 的 GUI 测试中的驱动事件。

状态（State）：即 Activity 界面中所有组件的状态集合，用户触发的驱动事件会引起组件变化或 Activity 间的跳转，即引起状态变化。

2.2 DFS 的测试模型

DFS 算法是一种基于模型的探索策略。在 DFS 的自动化测试过程中，主要的思想是用深度优先和回溯的策略来选择驱动事件，进行测试过程中 Android 状态的转移，形成一个状态转移图或有限状态机。在测试流程中，核心的算法流程包括驱动事件的获取，选择驱动事件的策略，状态转移图的构造以及状态的回溯等等。

首先是驱动事件的获取，在 Android 的 GUI 测试中，每个状态都有自己的界面，界面中包含了众多的界面元素，而每个界面元素的都包含了一个或者若干个可以触发的事件，如 TextView 上可以触发点击事件、EditText 上可以触发输入文本的事件和长点击事件、ListView 上可以触发滑动事件。在每个状态下获取该界面的所有的 GUI

元素，并且遍历 GUI 的所有元素，获取所有的驱动事件，构成驱动事件的集合 EventSet.

其次是选择驱动事件的策略，在现有的 DFS 算法中，根据当前获取到的驱动事件的全集 EventSet，选取第一个尚未被探索过的驱动事件，直到该状态下的所有驱动事件都被探索结束。其中，有两种情况，第一种情况是，在触发驱动事件之后，会进入到一个新的未被探索过的状态，则根据 DFS 算法的原理会深度优先地探索该新的子状态；第二，在该状态下的驱动都被探索结束，无需继续探索或者触发驱动事件之后进入一个曾经探索过的状态，则需要进行回溯，在 DFS 的探索算法中，回溯是一个十分重要的过程，将当前状态回溯到一个有效的祖先节点上。当状态 A 是状态 B 的祖先状态，从状态 B 回溯到状态 A，则需要重启应用回到应用的初始状态/根状态，然后执行初始状态到状态 A 之间的所有的驱动事件的序列，到达状态 A。

2.3 DFS 的测试中的问题

DFS 测试算法在理论上是遍历整个应用的所有状态下的所有的驱动事件，然后在实践过程中发现，该策略往往会显得效率低下、测试的覆盖率难以达到很好的效果。其根本原因在于在驱动事件的选择过程中，进行依次遍历所有的驱动事件，而在整个测试流程中，测试的时间是有限的，测试的状态又是复杂而且易变的，现在市场上主流的应用功能都很丰富，包含的界面数量以及界面的复杂性快速增加，依次遍历所有的驱动事件在有限的时间内已经不再是最有效最合理的选择。在 Android 测试过程中，由于界面的元素的不断变化和驱动事件带来的状态转移，测试过程将会不断产生新的状态，构成无限的测试状态，导致探索路径爆炸[11]。

其次，驱动事件的集合，将会有很多的冗余事件在其中，比如图 1 所示的 TextView“美食”字样和 ImageView 图片在触发点击事件之后，它们所引起的测试状态的变化是相同的，我们定义这样的驱动事件是冗余的驱动事件，其带来的影响一方面导致无效的状态转移，测试过程重复效率变低；另一方面是测试状态是进入一个曾经探索过的状态，在该状态需要进行回溯。回溯在 Android 测试算法中，需要重启应用，将应用的状态从根节点执行到需要回溯到的目标节点的路径，回溯重启应用是一个十分耗时的过程，在测试过程中应该尽量避免低效的回溯[11]。

Fig.1 Redundant Driven Event
图 1 冗余驱动事件

最后，针对 DFS 遍历过程中的驱动事件，例如文本输入的驱动事件，在许多的安卓自动化测试研究工作中，都回避了触发文本输入的驱动事件，或者是选择输入随机字符串的方式，希望能触发文本输入事件中可能存在的 Bug。但是在实际测试过程中，许多的文本输入事件需要特定的有效信息才能促成有效的 Activity 或者状态的跳转，如输入注册邮箱，现在 Android 市场上的很多流行软件在未登录状态下能遍历到的 Activity 只是该应用所有 Activity 的很少部分。

3 规则驱动下的 Android 测试

3.1 规则驱动的测试模型

测试 DFS 算法在测试过程存在着效率和覆盖率较低的问题，根据 2.3 节分析，我们发现依次遍历所有的驱动事情以及测试过程中的大量回溯过程是其根本原因。在测试的运行实践中，DFS 的自

动化探索过程是对测试的应用无先验知识的，对应用的遍历测试是试探性的；而导致低效的探索过程和回溯过程，在有先验知识的指导下，将会得到有效的避免。

```
算法 1: RDTA-Test
输入: 当前状态 currentState,上一个状态 parentState;
输出: 深度优先搜索树 DFSTree;
00: set widgetTree=GetUIElement(currentState);          //获取当前 State 的界面元素集合
01: set events = GetEventsByRuleEngine(widgetTree)      //规则引擎根据当前界面元素生成驱动事件集合
02: repeat
03:       set event=GetFirstUnfiredEvent(events);       //获取第一个未被执行的 event
04:       Fire(event);   //执行 event
05:       set newState=GetCurrentState();               //获取当前 State
06:       if IsLeaf(newState) then                      //判断该 State 是否需要继续探索
07:            BackTo(currentState);                    //回溯到上一状态
08:            continue;
09:       else
10:           DFSTree(newState, currentState);          //递归探索子节点
11:       end if
12: until AllFired(events)                              //直到该 State 所有 Event 已经执行完毕
13: if parentState==NULL then                           //判断是否为根节点
14:       return DFSTree;                               //测试结束，返回生成的 DFSTree
15: else
16:       BackTo(parentState);                          //回溯到上一个状态
17:       return NULL;
18: endif
```

Alg.1 GUI testing algorithm based on Rule-driven DFS

算法 1 规则驱动的 DFS 的 GUI 测试算法

测试人员的在特定的界面，对于该界面需要测试的界面元素、需要避免测试的界面元素以及对界面元素测试的优先级等等，往往有更好的理解；对于测试过程中 DFS 自动遍历算法中的难以处理的文本输入，对于测试人员而言却可能是简单的操作。如果通过将测试人员的先验知识注入到自动化测试过程中，约束和限制 DFS 的依次遍历所有驱动事件的方式，将会达到对整个测试的 DFS 树约简的作用，避免很多无效冗余的驱动事件，避免十分耗时的回溯过程以及重启应用。

因此，对于 DFS 的自动化遍历过程，需要提供一种用户干预的机制，用户通过配置外部规则的方式将其先验知识，注入到自动化的测试过程中，DFS 的测试遍历过程将不会是完全不受限制的，而是受规则约束的，即是规则驱动的遍历测试。规则起作用是在生成和选择驱动事件的时候，约束所有驱动事件的集合或者对驱动事件进行定制。

3.2 规则的内涵

规则的作用是将测试人员的先验知识用规则的方式来驱动测试的自动化过程，因此规则的目标是针对当前的 Activity 下的界面元素和驱动事件的集合，通过规则引擎的驱动，能够提高 DFS 测试的效率，避免无效的驱动事件，优先选择有效的能引起状态转移的驱动事件，以较低优先级触发无效的驱动事件，从而提高有限时间内测试的覆盖率。

DFS 默认的遍历策略是依次遍历所有的驱动事件，规则是针对“依次”、“所有”和驱动事件三个方面。通过规则来从驱动事件的全集合中选择出一个有效的子集合，并且对驱动事件的优先级和实际操作如点击、滑动、文本输入内容等有一定的约束，根据测试的实践过程和测试人员的先验知识，往往会尽快触发能够引起状态转移的驱动事件，期望能在有限的测试时间内达到尽高的覆盖率。规则的语义包含以下若干方面：

1、界面元素的子集合：规则定义的单元，可以是某个特定的界面元素或者某一类相似的界面元素。

2、子集合的遍历方式：该集合下的界面元素的个数可能很多，规则可以指定遍历的方式和限

制，如指定顺序遍历、逆序遍历、随机遍历和遍历次数限制等，如设置遍历次数的限制可以避免在遇到状态爆炸时的问题。

3、子集合上的操作事件：通过指定操作事件，可以更好地进行测试的过程，如对于文本输入的界面元素可以指定输入正确的文本，对于需要滑动的界面元素也可以指定滑动方向。

4、子集合的优先级：优先级是相对于其他子集合的，优先级可以使得引起状态转移和 Activity 跳转的驱动事件优先被触发，而低效冗余的被完全剔除或者优先级很低。

4 规则驱动的实现

4.1 规则的表示

规则的实现方式是一个外部定义规则的文件以及在选择驱动事件时的对应规则验证引擎。

每条规则都是一个五元组<activity> <selector> <modifier> <action> <priority>, 其中 activity 表示规则所约束的活动名；selector 表示界面元素子集合的选择器；modifier 表示对当前选择器的修饰器；action 表示对界面的元素的具体操作，对界面元素的操作即驱动事件；priority 表示在当前状态下该选择器的界面元素驱动事件在所有驱动事件集合中的优先级。

活动名是在一个应用能够定位到特定界面的信息，如<me.ele.Launcher>表示饿了么应用的主界面的 Activity。

选择器的作用是从测试过程当前状态的界面元素的集合中，选择出需要制定规则的子集合，该集合中元素的个数可以是某一个特定的界面元素，在当前界面所有的元素集合中是唯一的，如在图 1 在有字样“美食”的 TextView 是唯一的；也可以是某一类界面元素，这一类元素具有某些共性，如 class 都是 ImageView。界面元素是我们规则约束的主体，是用来定位特定的 Widget 元素在 Android 的 GUI 层次结构树上位置的信息。

修饰器的作用是进一步约束该界面元素的子集合上的遍历方式，如图 2 对于一个列表下会有很多表项，而每个列表项的遍历过程是相对等价的，如果遍历所有的列表项则会使用大量的时间和重复遍历旧的状态，因此可以通过选择遍历的策略和次数限制选择若干个列表项进行抽样测试。

Modifier	规则语义
Random(value)	随机遍历
TopFirst(value)	顺序遍历
BottomFirst(value)	逆序遍历
注：value 表示遍历的次数限制	

Table.1 Modifier
表 1 修饰器

操作器的作用是约束选择器中界面元素集合上的操作，其可选值可以是表 2 中的任意值，对界面元素上的操作即构成了特定的驱动事件。如对于 TextInput 操作可以自定义输入的文本信息。

Action	操作语义
Click	点击
LongClick	长点击
Swipe Up/Down	向上/下滑动
Swipe Left/Right	向左/右滑动
TextInput(Value)	输入文本域

Table.2 Action
表 2 操作器

优先级的作用是用来设置界面元素子集合的优先级，默认的优先级为 0，优先级大于 0 的界面元素表示被拉入白名单，其上的驱动事件将会优先被触发；而优先级的值可以设定为-1，表示该类界面元素被拉入黑名单，该类界面元素上的驱动事件是冗余的，往往会导致进入旧的状态引起回溯。

Fig.2 The ListView
图 2 列表视图

4.2 选择器的实现

在 Android 的 GUI 模型中，每个 Activity 界面结构都是一棵分层的树结构，在界面组件的外层一般都存在 LinearLayout、FrameLayout 等布局元素，如图 1 中的 ImageView 和 TextView 被包含在 LinearLayout 里面。

UIAutomatorViewer 是 Android SDK 包中原生的开源工具，可以获取界面的详细信息，给用户提供一种查看当前界面布局、控件属性的工具，用户可以获取当前界面层次结构树的信息，并保存在 XML 文件中。

```
<!--父节点-->
<node index="0" text="父节点" resource-id="android:id/decor_content_parent"
  class="android.view.ViewGroup"
  package="com.zte.heartyservice"
  content-desc=""
  clickable="false" >
    <!--子节点-->
    <node index="0" text="子节点 1"
      resource-id="android:id/bar_con tainer" />
    <node index="1" text="子节点 2"
      resource-id="android:id/content" />
</node>
```

Fig.3 XML GUI Tree
图 3 XML GUI 树

其中，每个界面元素包括布局元素和其他界面元素都是界面树中的一个节点，节点信息包含了该界面元素的所有属性信息，所有的属性信息既可以用来标识一个元素的特性，也可以根据上面的功能性属性来生成特定的驱动事件，包括以下如 text，resource-id, class, package, contect-desc 等标识性属性，以及 clickable, long-clickable, checkable 等功能性属性。

Selector 选择器的实现就是基于控件的属性信息，以及 GUI 树的祖先后代关系、兄弟关系等。利用控件自身的属性信息可以包含 XML 中如图 3 所示所有的详细信息，如 class 和 text 信息；利用 GUI 树的关系描述可以有效的利用节点与节点之间的关系，如图 1 中例子，TextView 节点和 ImageView 节点都是 LinearLayout 的子节点；而且 TextView 和 ImageView 又是兄弟关系。

在 Selector 选择器中，我们需要能够在保存着界面层次树的 XML 文件中利用元素的属性和元素之间的关系来选择元素的集合。XPath 语言是一门在 XML 文档中查找信息的语言。XPath 可以通过元素和属性对节点进行选择。对应选择器中，搜索的节点是界面的 Widget 元素，而搜索的空间是关于界面的层次结构树的 XML 文件。

XPath 的路径表达式可以支持很多谓词、通配符和运算符，使得它可以很方便的定位到需要的节点。而且，XPath 适合表示界面组件的层次树结构的关系。如图 4 中所示的 XPath 路径选择表达式，selector1 表示选取所有 class 属性为 TextView 的界面元素； selector2 表示选取 text 内容为美食的界面组件；selector3 表示选取父节点为列表 ListView 下的 ImageView 界面元素，利用到了节点之间的层次关系。

```
{"selector1": "//node[@class='an-
droid.widget.TextView']"}

{"selector2": "//node[@text='美食']"}

{"selector3": "//node[@class='an-
droid.widget.ListView']//node[@class='an-
droid.widget.ImageView']"}
```

Fig.4 Examples of Selector
图 4 选择器示例

4.3 规则验证引擎的实现

规则引擎是实现外部规则驱动自动化测试过程的引擎。

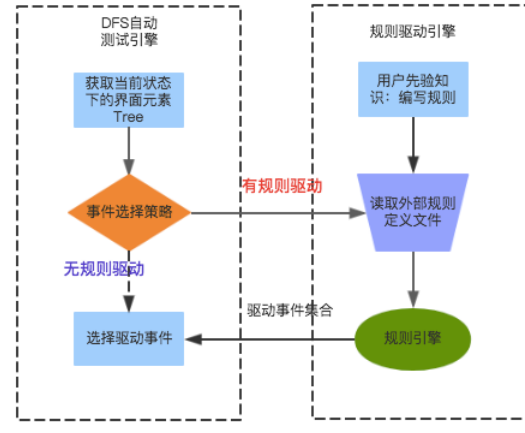

Fig.5 The Rule-driven Model
图 5 规则驱动模型

被测应用	Activity 个数			Bug 个数		
	DFS	RDTA	Monkey	DFS	RDTA	Monkey
手机管家	8	35	10	0	0	0
日历	7	8	5	0	0	0
联系人	7	11	8	0	0	0
客服	14	43	11	0	0	0
百度外卖	4	10	7	0	0	0
糯米	5	7	5	0	0	0
饿了么	15	25	12	1	1	0
有道词典	6	15	7	0	1	0
酷我音乐	3	7	3	0	0	0
美颜相机	14	27	12	0	0	0
熊猫 TV	7	20	8	0	1	0
百度地图	5	10	2	0	0	0

Table.3 Activity and Bug Counts of different Test Methods
表 3 不同测试方法的活动和异常个数结果

图 5 是 DFS 自动化测试在有规则驱动和无规则驱动的流程，默认情况下的 DFS 会根据当前界面的 GUI 树生成全部的驱动事件的集合，该集合中驱动事件在没有规则的约束下，会有很多冗余的驱动事件，并且没有优先级关系，只能依次遍历所有的驱动事件。如算法 1 所述，规则驱动引擎根据当前的界面的元素，通过外部规则的选择器选出子集合，对子集合上元素定义了遍历的顺序，驱动事件的具体操作，该子集合在全集合中的优先级。通过规则驱动的测试的驱动事件，性的、有序有优先级的驱动事件集合。

5 实验结果与分析

基于本文提出的算法，我们设计了测试工具 RDTA（Rule-driven Testing Platform for Android），并且在真实的 Android 应用上进行了实验的验证。

我们使用的实验设备是 Nexus 5，Android 版本为 6.0.1，我们选取了 12 个来自 Android 市场上的实际应用。其中对于每个应用我们做了对比实验，每组实验时间限制为 1 个小时，实验中对比的测试技术分别是 Monkey 随机测试、无规则驱动的 DFS 测试技术（以下简称 DFS）和有规则驱动的 DFS 测试（以下简称 RDTA）。根据[7]中对现有测试技术的实验对比，使用 Monkey 随机测试通用性和效果较好，因此具有对比性，而通过是否有规则驱动的 DFS 遍历实验的对比，我们希望回答的问题是：

1、规则驱动的 DFS 遍历测试是否能提高测试的覆盖率和效率；

2、规则驱动能够提高测试覆盖率和效率的根本原因；

3、编写外部规则的易用性。

被测应用	DFS			RDTA		
	NEW	OLD	OLD /NEW	NEW	OLD	OLD /NEW
手机管家	26	98	3.8	57	122	2.1
日历	23	192	8.3	25	178	7.1
联系人	39	163	4.2	67	118	1.8
客服	53	192	3.6	92	102	1.1
百度外卖	59	102	1.7	81	114	1.4
糯米	40	82	2.1	51	75	1.5
饿了么	40	121	3.0	85	64	0.8
有道词典	34	161	4.7	106	316	3.0
酷我音乐	29	321	11.1	70	234	3.3
美颜相机	46	152	3.3	62	136	2.2
熊猫 TV	23	81	3.5	73	160	2.2
百度地图	32	120	3.8	50	107	2.1

Table.4 State
表 4 状态

对于问题 1，分析表 3 中实验结果，无规则驱动的 DFS 遍历 Activity 的数量有 6 个应用比随机 Monkey 效果较优，有 4 个应用比 Monkey 较差,有 2 个应用相同；而 RDTA 测试过程中的 Activity 覆

盖率普遍比随机的 Monkey 和 DFS 高，并且随着测试覆盖率的提高，RDTA 在测试过程中也发现了较多的 Bug。

对于问题 2，我们在实验过程中对比了 DFS 和 RDTA 测试过程的状态转移情况如表 4 所示，其中状态类型为 NEW 表示该状态在测试过程中是未探索的的情况下被第一次探索到的，即发现了一个新的状态；而 OLD 表示该状态在之前的探索过程中已经被访问过了，在该状态下需要进行状态的回溯，降低测试的效率和覆盖率。从实验结果可以看出，RDTA 相较于 DFS 的状态，能够发现较多的 NEW 状态，而避免进入 OLD 状态引起回溯。降低了 OLD/NEW 的比例，就是 RDTA 相较于 DFS 测试效果变好的根本原因。

对于问题 3，我们对饿了么应用的规则进行简单阐述，该应用的规则只是简单的写了 5 条，测试的效率就有了明显的提高。例如，规则 1 和 2 是针对 ImageView 和 TextView 上的驱动事件，将 TextView 的优先级设置为 1，ImageView 的优先级设置为-1，就可以减少触发 ImageView 和 TextView 上冗余的驱动事件；并且对于 TextView 的遍历方式我们采取的是顺序遍历 20 个 TextView 类型的界面元素，这样的限制可以避免状态爆炸的问题；对于 ImageButton，在实验中发现是回退的一个按钮，将其优先级设置为-1，可以避免一些低效的回溯；对于特定的.bwq 的 Activity 下是关于用户登录名和密码的，该类复杂事件自动测试是难以生成的，我们通过规则指定其输入为特定的值如“用户名”、“密码”。通过该示例可以发现，通过简单的若干条规则即可以提升较好的效果。

Activity	Selector	Modifier	Action	Priority
.Launcher	//node[@class='android.widget.TextView']	TopFirst(20)	Click	1
.Launcher	//node[@class='android.widget.ImageView']	Random	Click	-1
.Launcher	//node[@class='android.widget.ImageButton']	*	Click	-1
.bwq	//node[@text='手机/邮箱/用户名']	*	TextInput(用户名)	2
.bwq	//node[@text='密码']	*	TextInput(密码)	2

Table.5 Rule Example Based on the Eleme App
表 5 饿了么应用规则示例

6 结论

本文首先阐述了相关工作，对现有的各种 Android 应用 GUI 测试技术进行了概述，其中基于 DFS 的测试算法在测试技术中应用最为广泛；而脚本测试的方式是最适合表达应用测试人员的先验知识和测试路径约束。其次，本文总结了基于 DFS 算法的 Android 应用的 GUI 的重要概念，并且抽象了测试模型和总体框架。对 DFS 测试过程导致效率低下和覆盖率不理想的原因从不同方面进行了总结。因此，本文着重研究了规则驱动的 Android 应用测试技术，将 DFS 的自动化测试与测试人员的外部规则结合，以达到更好的测试效果。外部规则的制定是针对 DFS 自动化测试中的问题，根据不同的问题，制定了规则的语义内涵，并且外部规则实现具有简单和通用性的特点。最后，我们根据规则驱动的方案，设计了 RDTA 测试工具，验证了规则驱动的测试技术对测试的效率和覆盖率有了很大的改进，并且能够检测到可能存在的异常。

本文虽然针对 DFS 中存在的若干问题，使用规则的方式来约束完全自动化的测试过程，在测试效率上达到一些改进；然后，测试过程存在的问题远不止这些，我们的规则具有良好的扩展性，仍然可以添加很多其他语义的规则并且实现相应的规则验证的引擎；其次，本文实现的 RDTA 还不支持复杂的手势事件和其他复杂事件，如录制声音等等，导致某些测试状态不能进入，影响测试覆盖率；在很多状态下，依赖其他的状态，如查看饿了么历史订单则需要有订餐记录，否则测试难以覆盖。本文的未来工作将会扩展规则的语言，形成功能通用的

DSL(Domain Specific Language)语言，并且以低耦合的方式驱动测试过程，达到更好的测试覆盖率。

致谢 本文工作受中兴通讯研究基金资助，在此我们表示由衷的感谢。

References:

[1] Amalfitano D, Fasolino A R, Tramontana P, et al. Using GUI ripping for automated testing of Android applications[C]//Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM, 2012: 258-261.

[2] Amalfitano D, Fasolino A R, Tramontana P. A gui crawling-based technique for android mobile application testing[C]//Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on. IEEE, 2011: 252-261.

[3] Amalfitano D, Fasolino A R, Tramontana P, et al. A toolset for GUI testing of Android applications[C]//Software Maintenance (ICSM), 2012 28th IEEE International Conference on. IEEE, 2012: 650-653.

[4] Azim T, Neamtiu I. Targeted and depth-first exploration for systematic testing of android apps[C]//ACM SIGPLAN Notices. ACM, 2013, 48(10): 641-660.

[5] Mirzaei N, Malek S, Păsăreanu C S, et al. Testing android apps through symbolic execution[J]. ACM SIGSOFT Software Engineering Notes, 2012, 37(6): 1-5.

[6] Mahmood R, Mirzaei N, Malek S. Evodroid: Segmented evolutionary testing of android apps[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 599-609.

[7] Choudhary S R, Gorla A, Orso A. Automated test input generation for android: Are we there yet? [C]//Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. IEEE, 2015: 429-440.

[8] Li X, Jiang Y, Liu Y, et al. User guided automation for testing mobile apps[C]//2014 21st Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2014, 1: 27-34.

[9] Memon A M, Banerjee I, Nagarajan A. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing[C]//WCRE. 2003, 3: 260.

[10] Anbunathan R, Basu A. A recursive crawler algorithm to detect crash in Android application[C]//Computational Intelligence and Computing Research (ICCIC), 2014 IEEE International Conference on. IEEE, 2014: 1-4.

[11] Choi W, Necula G, Sen K. Guided gui testing of android apps with minimal restart and approximate learning[C]//ACM SIGPLAN Notices. ACM, 2013, 48(10): 623-640.

[12] Machiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for android apps[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 224-234.

[13] Z. Qin, W. Pan, Y. Xu, K. Feng, and Z. Yang, "An efficient scheme of detecting repackaged android applications, " ZTE Communications, vol. 14, no. 3, pp. 60–66, Aug. 2016.

附中文参考文献：

[14] 张灿，薛云志，陈军成．一种基于 Android 平台 GUI 录制回放工具的设计与实现[J]. 计算机应用与软件, 2012, 29(12): 6-9.

[15] 张大伟,郭烜,韩臻.安全可信智能移动终端研究[J].中兴通讯技术,2015,21(5):39-44