國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

Invariant Based Test Oracle Construction for Android Apps

陳宗堯

Tsung-Yau Chen

指導教授：王凡 教授

Advisor: Fang Wang, Professor

中華民國 104 年 7 月

July, 2015

國立臺灣大學碩士學位論文
口試委員會審定書

以不變量為基底的安卓應用程式測試準則

Invariant Based Test Oracle

Construction for Android Apps

本論文係陳宗堯君（R00943156）在國立臺灣大學電子工程學研究所完成之碩士學位論文，於民國 104 年 07 月 10 承下列考試委員審查通過及口試及格，特此證明

口試委員：

[illegible]

（指導教授）

陳明義　　陳銘憲

廖弘中　　雷欽隆

系主任、所長　劉深淵

中文摘要

自從智慧型手機問世以後，行動裝置的應用程式已經變成軟體產業中很重要的一部分。Android 是目前行動裝置平台使用者最多的。而現在有關 Android 應用程式測試的研究也變得更加的完整。然而，現在並沒有一個關於 Android 的測試準則方法可以用。測試準則是測試自動化當中重要的環節。大多數的應用程式都需要一個測試準則來驗證待測物的行為就是正確與否。現在大多數的晚體工程師都是用人力來做這樣的判斷，但是這樣的代價是大量的時間跟金錢成本。這就是為何現在測試準則已經逐漸變成一個自動化測試上的瓶頸。

這篇論文介紹這個系統可以利用不變量測試的概念去建構測試準則，應用的環境是黑箱測試。這個系統能抓取執行紀錄的特徵，用來建構測試準則。然後我們就能使用這測試準則來預判一個待測物究竟會是成功或失敗。我們的目標是希望能減少人力，同時盡可能地達到更高的精準度。

關鍵字：軟體測試，安卓應用程式測試，測試準則，不變量測試。

ABSTRACT

Since the invention of the smart phone, the mobile application becomes the important part of the software industry. Android has the most users in mobile platform. The researches about Android application testing technology are more and more completely. However there is no test oracle construction tool for Android. The test oracle is an important part of test automation. Most applications need the test oracle to verify the software under test for whether it behaviors correct or not. Recently, software engineers always do it by human power, which costs lots of time and money. It’s why the test oracle is becoming a bottleneck of the automated testing.

This paper introduces a system that constructs the test oracle by the concepts of invariant testing. It works on black-testing. The system gets the features of execution traces, build the test oracle up. Then we can use the test oracle to predict the behaviors of the software under test. Our goals are reducing the human cost, and getting as high accuracy as possible.

Keywords: software testing, Android application testing, test oracle, invariant testing

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1   Introduction

While the smartphone is becoming popular, hundreds of thousands of specialized applications, called apps, are available for these mobile platforms. The applications are getting more and more complicated. It is a big challenge for software testing engineers.

In this paper, we are focused on the test oracle construction for the Android applications. Test Oracle is an important part of software testing. It is a mechanism for determining whether a test case has passed or failed. In another word, with a given test case, the Test Oracle predicts what the result is. The construction of the test oracle is often much harder than it seems to be. It involves problems related to controllability and observability problem. It is the reason why the test oracle becoming the bottleneck of automated testing. Constructing the Test Oracle makes a great help for improving the testing efficiency.

We are trying to build the test oracle up for black-box testing. Because of we do not always know the specification of the applications. Even if we got the specification, the transform from the application specification to the test oracle always needs some human knowledge. The description of this transform is always fuzzy. We want a more reliable strategy to build the test oracle up for Android application testing.

We construct the test oracle by the concept of invariant testing. An invariant means a property that holds at a certain point or points in a program. For example, it includes *being a constant*($x = a$), *ordering* ($x < y$), *in a range* ($a < x < b$). The concept of invariant-based testing assumes that the application behaviors are consistent. An invariant should include three parts. First, what is the variable? Second, what is the grammar? Last, where is the program point it holds? Thus, invariants are the rule in the program point. After getting the constraints of each transition of the application, we can verify the testing cases, and then report our predicted results.

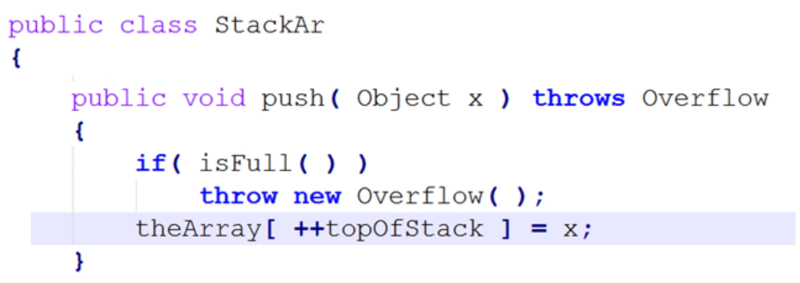
Chapter 2 Preliminary

2.1 Daikon

Daikon is an invariant detector tool implement by Java. It designed for detect invariants in C, C++, Java, Python and Perl programs. Daikon version 1.0 is published by Michael D Ernst at Massachusetts Institute of Technology.

The acceptable input format of Daikon is called "dtrace". The Daikon frontend program will execute the program and then output the variable information with dtrace format. Each language has its own frontend. It is the reason that Daikon can detect invariants for multi-language. Daikon detects the invariants of test case by reading the dtrace format files only. The format will describe the metadata of the variables and the value of each variable.

What the Daikon concerned is every procedure in the test case. Daikon detects the invariant according to the information that the dtrace file describes. Daikon will output the value with the grammar that holds in each procedure. We show a java program example below. It is the *push* function of the *stack* program:

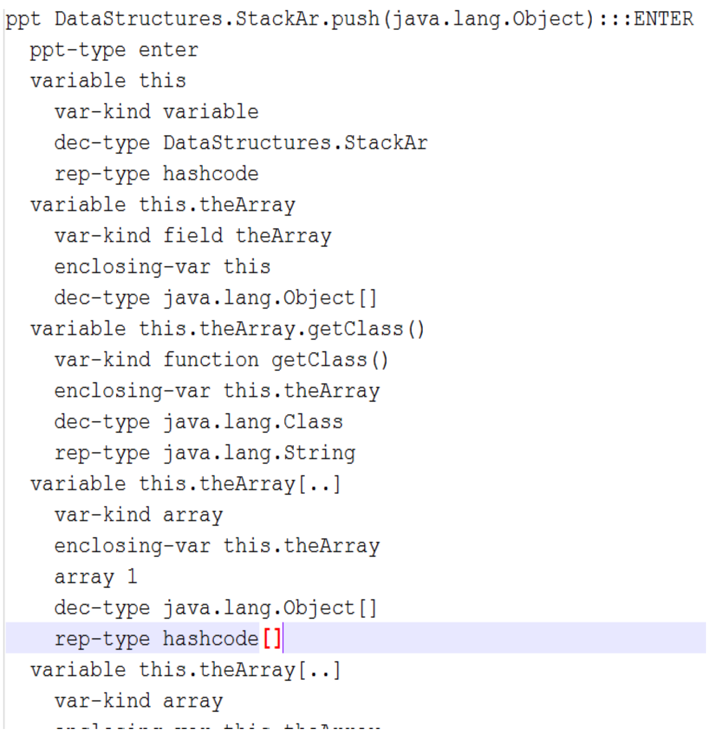


```
public class StackAr
{
    public void push( Object x ) throws Overflow
    {
        if( isFull( ) )
            throw new Overflow( );
        theArray[ ++topOfStack ] = x;
    }
```

Figure 2-1Push function

There are some examples of the Daikon input format as following.

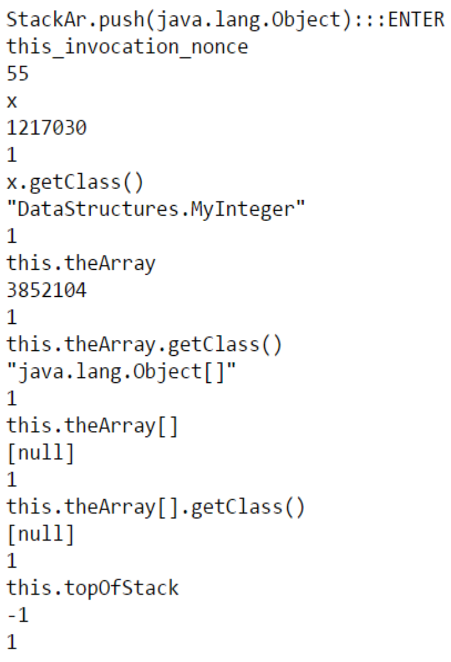
The top-half of the dtrace file is “program point declaration block”.



```
ppt DataStructures.StackAr.push(java.lang.Object):::ENTER
  ppt-type enter
  variable this
    var-kind variable
    dec-type DataStructures.StackAr
    rep-type hashcode
  variable this.theArray
    var-kind field theArray
    enclosing-var this
    dec-type java.lang.Object[]
  variable this.theArray.getClass()
    var-kind function getClass()
    enclosing-var this.theArray
    dec-type java.lang.Class
    rep-type java.lang.String
  variable this.theArray[..]
    var-kind array
    enclosing-var this.theArray
    array 1
    dec-type java.lang.Object[]
    rep-type hashcode[]
  variable this.theArray[..]
    var-kind array
```

Figure 2-2 top half of dtrace file

The bottom-half of the dtrace format is “value declaration block”



```
StackAr.push(java.lang.Object):::ENTER
this_invocation_nonce
55
x
1217030
1
x.getClass()
"DataStructures.MyInteger"
1
this.theArray
3852104
1
this.theArray.getClass()
"java.lang.Object[]"
1
this.theArray[]
[null]
1
this.theArray[].getClass()
[null]
1
this.topOfStack
-1
1
```

Figure 2-3 bottom half of dtrace file

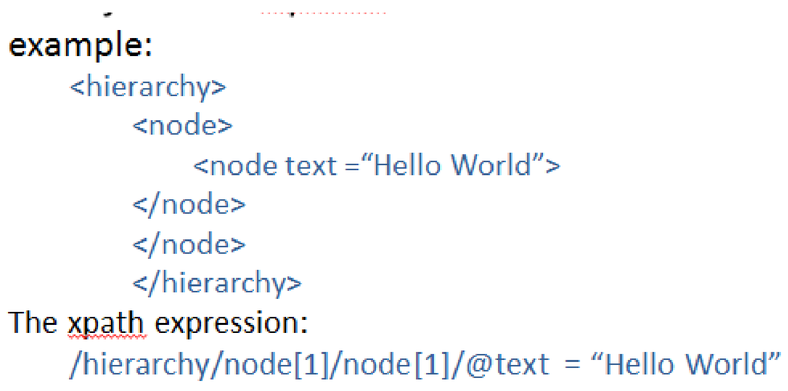
```
DataStructures.StackAr.push(java.lang.Object):::ENTER
x != null
=========================================================
DataStructures.StackAr.push(java.lang.Object):::EXIT
orig(x) == this.theArray[this.topOfStack]
this.topOfStack >= 0
this.topOfStack - orig(this.topOfStack) - 1 == 0
```

Figure 2-4 invariant report example

There is an “object block” between program point declaration block and value declaration block. Because we don’t need it for our purpose, we ignore it in this paper.

2.2 Xpath

Xpath is a query language for selecting nodes and attribute from a XML document.



example:

```
<hierarchy>
    <node>
        <node text =“Hello World”>
    </node>
    </node>
    </hierarchy>
```

The xpath expression:

    /hierarchy/node[1]/node[1]/@text = “Hello World”

Figure 2-5 example of xpath

And we modify the xpath expression as below:

/hierarchy/node[1]/node[1]/@text = “Hello World”

➔hierarchy.node[1].node[1].text = “Hello World”

Chapter 3 Framework

3.1 Overview

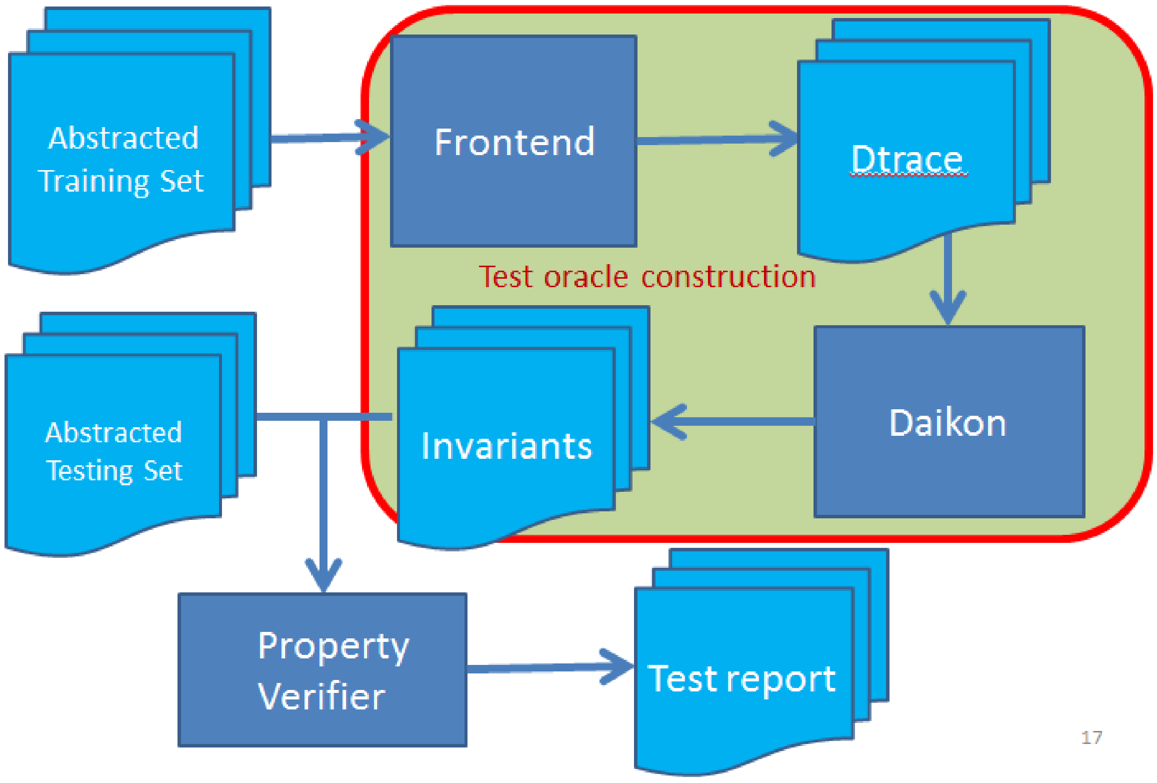


Figure 3-1 workflow graph

Component

- Test Oracle Construction
  - Frontend
    - ✧ Transform from xml file to dtrace format file
  - Daikon
    - ✧ Invariant detector
- Test Oracle execution
  - PropertiesVerifier
    - ✧ read constraints and input traces for verification

Abstracted data

In order to avoid the problem of the state explosion, we need to abstract the screenshot data. If we don't abstract the data, the test oracle can be only used for the same screen with the same action. And it will cause the number of states grow up prohibitively. It is the main reason why we need to abstract the screenshots and the actions.

Example of abstract mapping :

Assume we got abstracted set data

View : state0 : screen1, screen2 , screen5

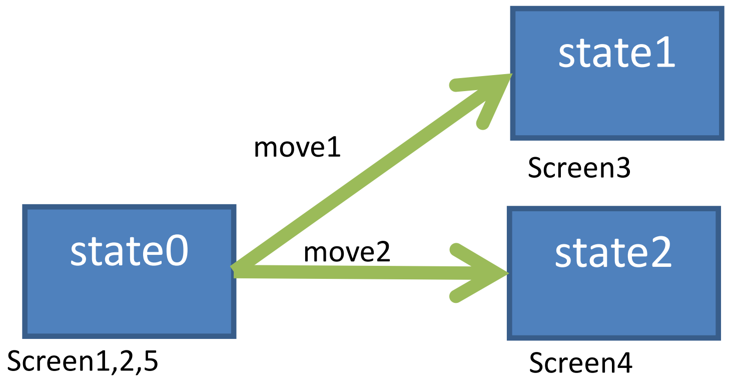
state1 : screen3

state2 : screen4

Edge : move1 : screen1 ➔ screen3

move2 : screen5 ➔ screen4

…



3.2 Concepts of using Daikon for GUI testing

Daikon focus on every procedure, then the Daikon find out the variables value in the precondition and the post-condition and report the invariants of each procedure. By this way, we use this concept in application testing. Actions → procedures in SUT, the source screen of an action → precondition of an SUT procedure, and the destination screen of an action → post-condition of an SUT procedure. By this way, we can convert the screen transition which is mapping to a program procedure.

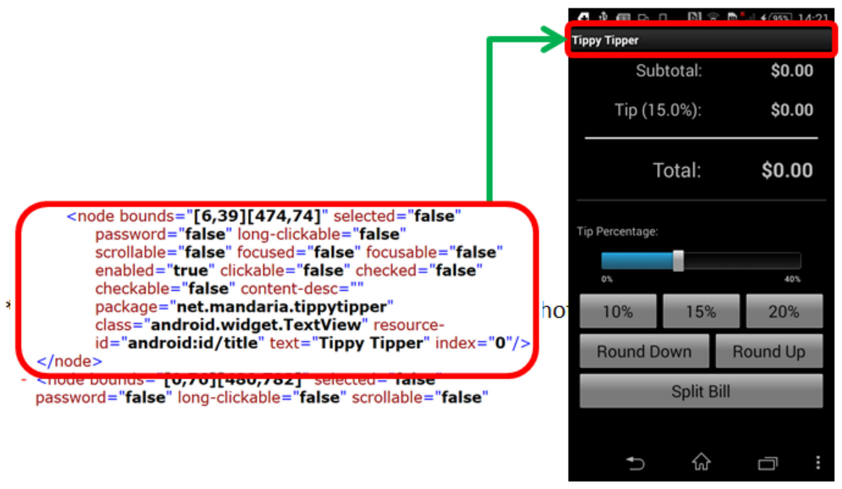
Dump xml files of screenshots

UIAutomator is a great GUI (Graphic User Interface) ripping tools has been published by Google. UIAutomator is a framework which is used to manipulate the UI form code into a Junit test case. UIAutomator is able to do some actions on a device such like: touch, scroll, take a screenshot, and ripping screen as xml file etc.

Our Test Oracle is based on the invariants of the xml attributes, and because of the UIAutomator only supports Android version 4.1 and after. If your SUT(software under test) is early version, you should find out another way for GUI ripping.

We can obtain screen data as XML file type by UIAutomator. So we can focus on any attribute in the XML type file. We can easily express the attributes and the values in Xpath expression.

Example of Daikon for GUI testing



```
    <node bounds="[6,39][474,74]" selected="false"
        password="false" long-clickable="false"
        scrollable="false" focused="false" focusable="false"
        enabled="true" clickable="false" checked="false"
        checkable="false" content-desc=""
        package="net.mandaria.tippytipper"
        class="android.widget.TextView" resource-
        id="android:id/title" text="Tippy Tipper" index="0"/>
</node>
- <node bounds="[0,76][480,782]" selected="false"
  password="false" long-clickable="false" scrollable="false"
```

Figure 3-2 xml & screenshot example(1)

The red block of the XML area is describes the red block of the screenshot, by this way we can get the attributes of the XML, which is mapping to data of the screenshot.

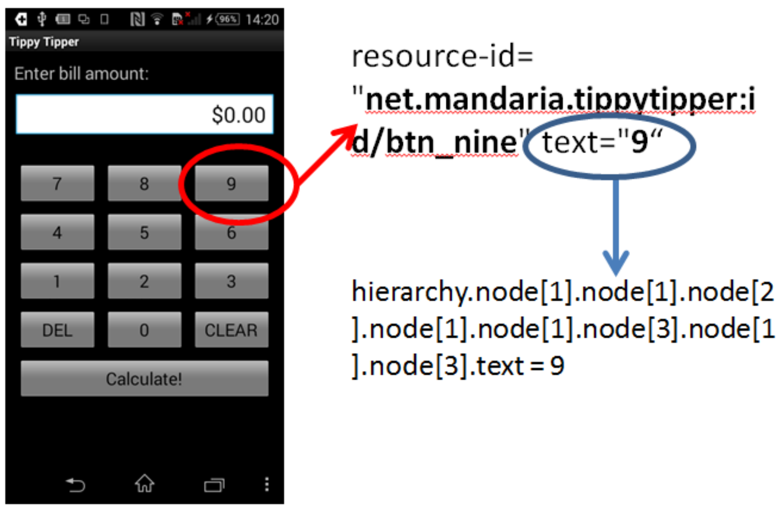

14:20
Tippy Tipper
Enter bill amount:
$0.00
7
8
9
4
5
6
1
2
3
DEL
0
CLEAR
Calculate!
resource-id= "net.mandaria.tippytipper:id/btn_nine" text="9"
hierarchy.node[1].node[1].node[2].node[1].node[1].node[3].node[1].node[3].text = 9

Figure 3-3 xml & screenshot example

There is a example of transform from xml attribute to xpath expression.

Reduce the complexity

Because of the Daikon invariants detect algorithm complexity is O(n^9) (n : the number of all the variables). It is a big problem for testing some larger size application. So we are necessary to reduce the number of the variables for solving the complexity problem.

1. **Filiter** :

   We can focus on the attributes we concern only. It will help reduce the number of attributes effectively.

2. **Fixed value**s

   For each abstracted state, we try to find out the variables with fixed values. so we can add those attributes in the constraints of the xml, so we are not need to describe in dtrace anymore.

3.3 Dtrace format constraints

Daikon is the best invariant detection tool as we known. Daikon has complete user manual, exception handler, and troubleshooting. Daikon designed for detecting normal programs. However, we treat Daikon as an XML invariant detector, which results in some additional problems while doing our experiments. Daikon makes some constraints checking which is reasonable for program, but not for XML transitions.

The following paragraph discuss about the extra constraint we met, and how to solve those constraints:

1. **Procedure variable check**

The Daikon checks that every variable in the pre-condition should also exist in the post-condition. In order to pass through this check, we modify the variable declaration rules. For any transition, the pre-condition and the post-condition, we declare both the variables in pre-condition and post-condition.

For example, there is a procedure from the screen A to the screen B, and the screen A has three attributes: a, b and c. And screen B has three attributes also, x, y and z. while we describe about the procedure, we should describe that there are six attributes in pre-condition and there are six attributes in post-condition too, they are a, b, c, x, y and z. However, in the value declaration of screen A, the attributes variable values which belong to the screen B would be an empty string. Similar to the screen B, the variable values of those attributes belongs to screen A would be an empty sting too. We can pass the procedure variable check by the modification on the declaration.

The weakness of this method is that we will increase the number of the variables, and it will make the loading of Daikon becomes heavier

2. Modify xpath expression

Because of the variable check in statement above, it will confuse us about attributes of XMLs. In order to solve this problem, we modify the xpath expression, we added the abstracted ID in front of every attributes in the xml files.

For example, there is a xpath is "a/b/@c", and the abstracted state of the attribute screen ID is "n". Then we change it to "n.a.b.c". So we can change the xpath expression by this way and no more confuse problem.

3. Undetermined state

There are some “move” actions which are different if the conditions are different. It means the same source state screen with the same move action, but it goes to another destination screen state. The situation is impossible for program procedure. While Daikon detects it happened, Daikon will report it as an error and reject the file. It is a big problem for detecting invariants.

In order to avoid the problem, when we got the traces set, we will build the undetermined state table up. When we declare the procedure, we list all the variables of the pre-condition and possible post-condition, like statement 1. It solves the format problem, so we can verify whether the transition behaviors following the invariants or not. Just like statement 1, the weakness of the solution is enlarging the number of variables.

4. check the variable type

In our purpose, we want to find out as much meaningful invariants as possible. It will increase the effectiveness and accuracy, and it will be easier for human to recognize.

The invariants of number relation are more powerful than invariants of string relation, so we want to support integer, floating point and more types. In order to avoid the dtrace format constraints, we check every value in each abstracted state. Find out the attributes with same type and build the table. While we are writing the dtrace file, we will declare the attributes with the corresponding type to improve the power of invariant detection.

In our experiment, we find that for a Calculator application, the text block write down the number calculated. But we cannot declare the block value as integer or floating point, because of there is sometimes be the "scientific notation".

Chapter 4 Algorithm

4.1 Test Oracle Construction

Input : Test cases with abstracted mapping data

1. Construct the table that describe how the screenshot XMLs mapping to abstracted states.
2. Compare every transition, find the undetermined transition out
3. Build undetermined table for describe the destinations of each undetermined state
4. Transform every xml into xpath type data. (attribute filter here)
5. Check the type of every attribute in each abstracted state.
6. For each abstracted state, check every attribute of every xml, find out the attributes with fixed value in the abstracted state.
7. For each transition:
   - Look up the correspond abstracted state.
   - Load precondition and postcondition attributes
   - Check the contents which is not contradicted to dtrace format constraints.
8. Output dtrace file by dtrace file format.
9. Daikon generate the invariant report.

4.2 FixedAttributeFinder :

Input : (name, value) list for all screen XMLs

```
OwnList = [ ]        # list all attribute with the value used before
FixedList = [ ]      # list all the attribute with fixed value

For each abstracted state s:
    for each xml in s:
        for each candidate in xml:
    if (the candidate is not in OwnList):
        OwnList.append(candidate)
        FixedList.append(candidate)
    else:
        if OwnList[candidate] != candidate :
        FixedList.remove(candidate)
Return FixedList
```

4.3 ProperVerifier

Input : traces set & invariant constraints

```
for each transition in SUT:
    for each Attributes in precondition, postcondition:
        If not ComplyConstraint(Attribute, constraints):
            Report (Attribute, constraints)
```

Chapter 5 Experiment

5.1 Experiment environment

• CPU	:	Intel® Core ™ CPU 2.5GHz *2
• Ram	:	16GB
• OS	:	Windows 64bit
• VM	:	VMware Workstation 8.0
• WM Ram	:	4GB

5.2 Benchmark

SUT1 : Trippy Tripper (17 View States):

It is a simple and open source Tip Calculator, it calculates the bill and the select tips, and split bill. Only 17 state under our testing. It is a really simple application.

Data: 104 pass, 96 fail traces, totally 200 traces.

SUT screenshot example as below

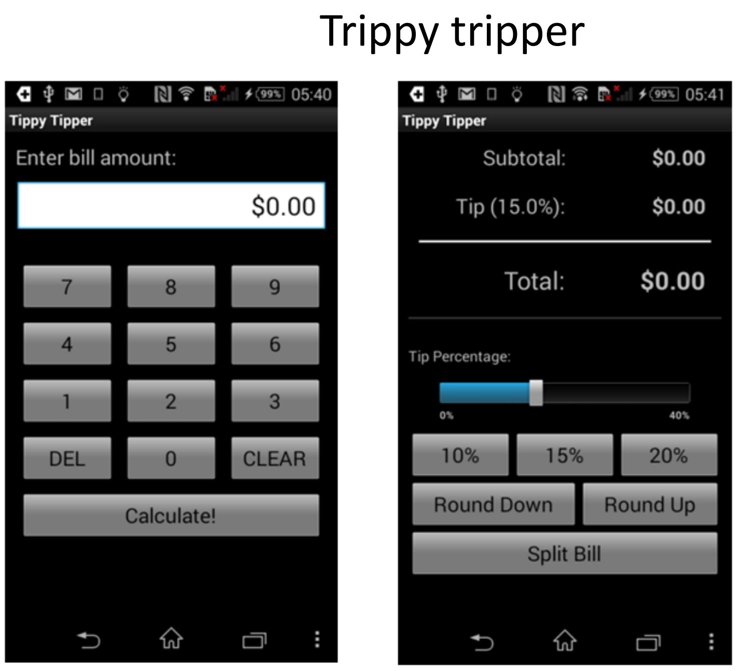


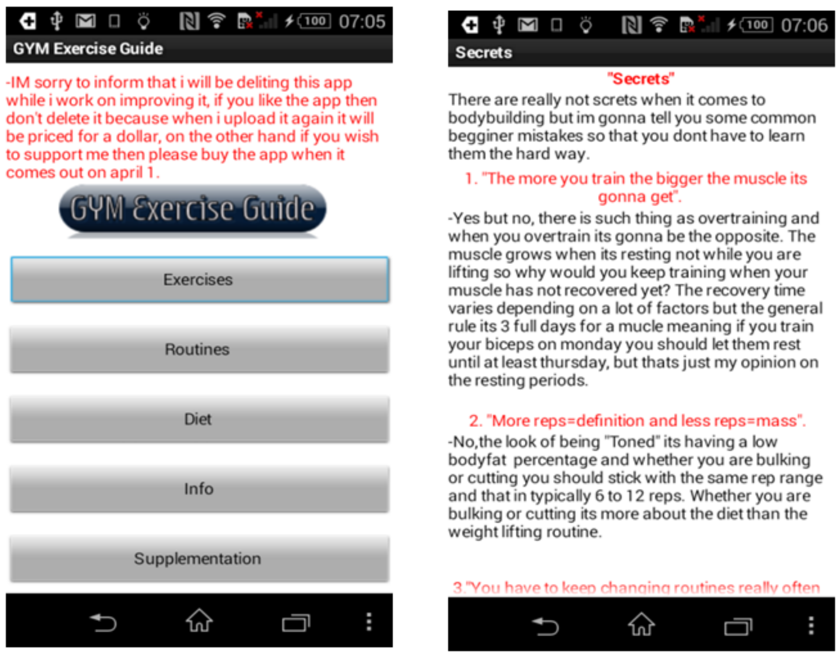
Figure 5-1 SUT:tippy tipper

SUT2 : GYM Guide(166 View States)

It a dictionary of GYM nouns, show the knowledge you should know before GYM exercises. It provides the suggestions for each GYM exercise, demo them and gives a simple exercise plan.

Data : 75 pass, 25 fail traces, totally 100 traces

GYM


07:05

GYM Exercise Guide

-IM sorry to inform that i will be deliting this app while i work on improving it, if you like the app then don't delete it because when i upload it again it will be priced for a dollar, on the other hand if you wish to support me then please buy the app when it comes out on april 1.

07:06

Secrets

"Secrets"

There are really not screts when it comes to bodybuilding but im gonna tell you some common begginer mistakes so that you dont have to learn them the hard way.

1. "The more you train the bigger the muscle its gonna get".

-Yes but no, there is such thing as overtraining and when you overtrain its gonna be the opposite. The muscle grows when its resting not while you are lifting so why would you keep training when your muscle has not recovered yet? The recovery time varies depending on a lot of factors but the general rule its 3 full days for a mucle meaning if you train your biceps on monday you should let them rest until at least thursday, but thats just my opinion on the resting periods.

2. "More reps=definition and less reps=mass".

-No,the look of being "Toned" its having a low bodyfat percentage and whether you are bulking or cutting you should stick with the same rep range and that in typically 6 to 12 reps. Whether you are bulking or cutting its more about the diet than the weight lifting routine.

3."You have to keep changing routines really often

Figure 5-2 SUT: GYM

5.3 Experiment Design

Experiment 1

- In order to test the accuracy of our test evaluation, we calculate the hit rate of the testing set with different size of the training data.
- We record the hit rate from 10 traces as training data to 50 traces as training data.

Experiment 2

- In order to find out the bottleneck of the test oracle accuracy, we calculated the hit rate of a larger testing set with fixed size of the training set.

- We analysis the result, and try to improve the test oracle.

Before we discuss about the experiment results, what we most concerned is the hit rate of the test oracle. The “pass” in our definition means that for each trace in traces set, every transition of the trace comply with the constraints, concluding pre-condition, post-condition and the relation between them. If any transition of the trace not complies with the rules, then we report the trace fail.

5.4 Experiments

False positive & false negative

In binary classification testing, there are two kinds of errors, False positive and False negative.

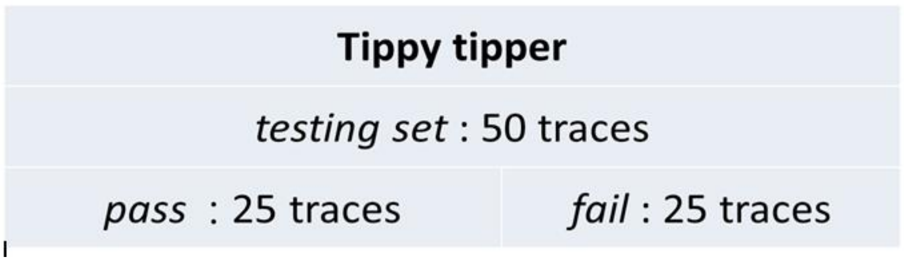
A false positive is an error in data reporting in which a test result improperly indicates presence of a condition. Such as a fail (the result is positive), when in really it is not.

A false negative is an error in which a test result improperly indicates no presence of a condition (the result is negative).

There are two kind of errors in a binary test, and are contrasted with a correct result

Experiment 1 :

Environment: Each SUT has 50 traces as max training data.



Tippy tipper	
testing set : 50 traces	
pass : 25 traces	*fail* : 25 traces

Table 1 Experiment1 : SUT 1

GYM guide	
testing set : 50 traces	
pass : 25 traces	*fail* : 25 traces

Table 2 Experiment1 : SUT 2

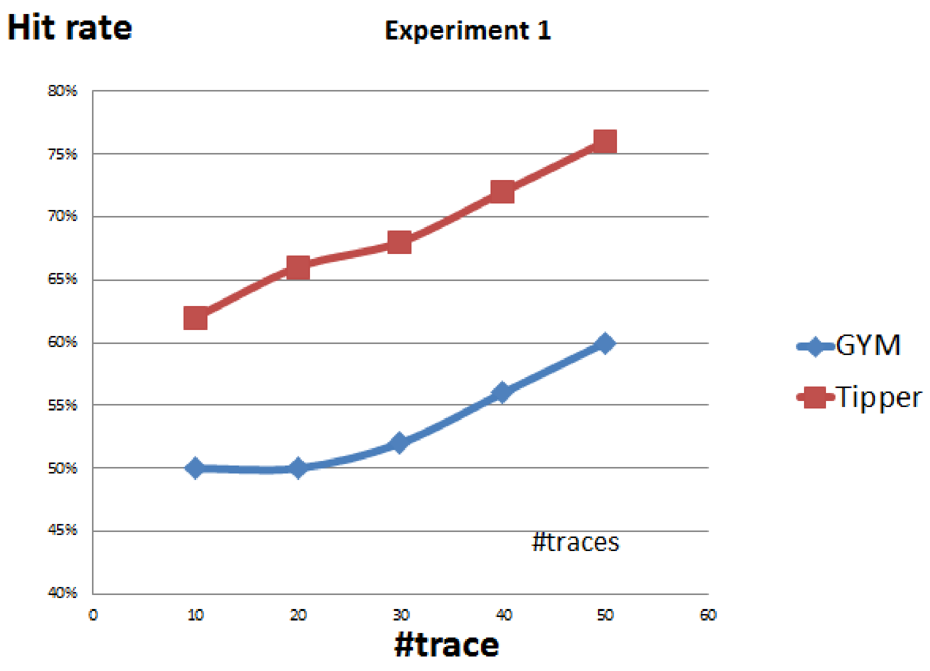


Figure 5-3 Test Oracle hit rate graph

In Experiment 1, the accurate of test oracle is raise up with the number of traces as training data. The application with lesser states "tippy tipper", hit rate is higher than the application with more states "GYM guide" obviously. But the hit rate is lower than what we respected. We will discuss what the problem is in experiment 2 result.

Experiment 2:

Environment:

The size of training set of each application is 50 traces

Tippy tipper	
testing set : 90 traces	
pass : 54 traces	*fail* : 36 traces

Table 3 Experiment2 : SUT 1

GYM guide	
testing set : 50 traces	
pass : 25 traces	*fail* : 25 traces

Table 4 Experiment2 : SUT 2

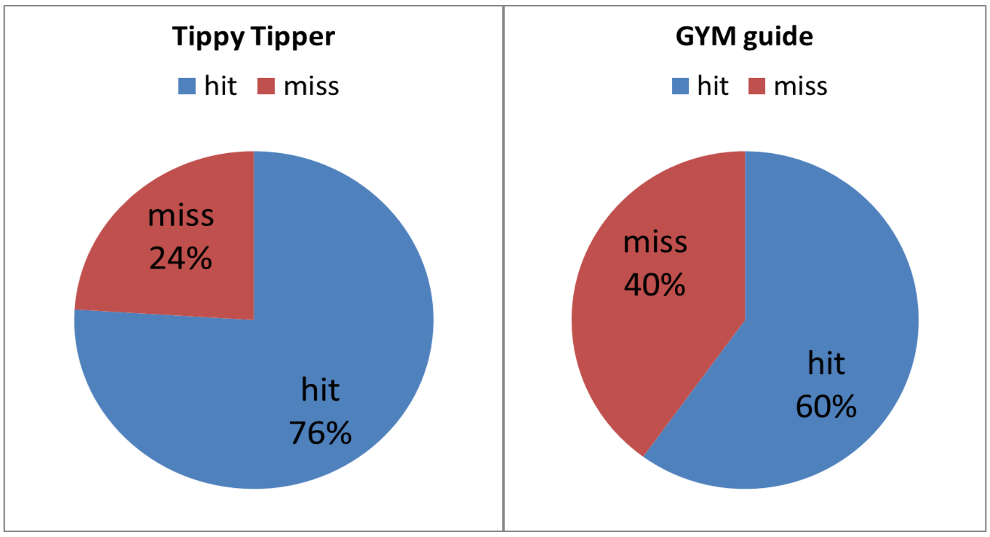


Figure 5-4 Experiment 2 hit rate

The hit rate is lower than our respect too. So we are trying to find out the reason by analysis the fail trace reports.

According the test case report, 100% of fail trace in our trace set are found out by test oracle. So there is no false negative problem in our experiment environment.

The problem happens in false positive type. Because of our "pass" and "fail" definition, we will report fail if the transition that is not included in the given training data. Somehow the transition might be reasonable transition. It is the reason why we are rarely found false negative result out (zero in our experiment).

Experiment 2:

Tippy tipper

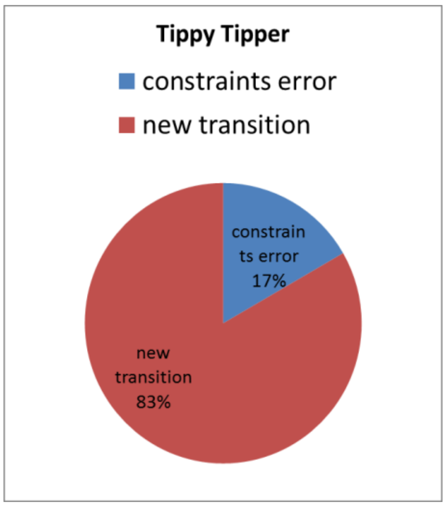


Figure 5-5 Experiment 2 error analysis (SUT1)

GYM guide

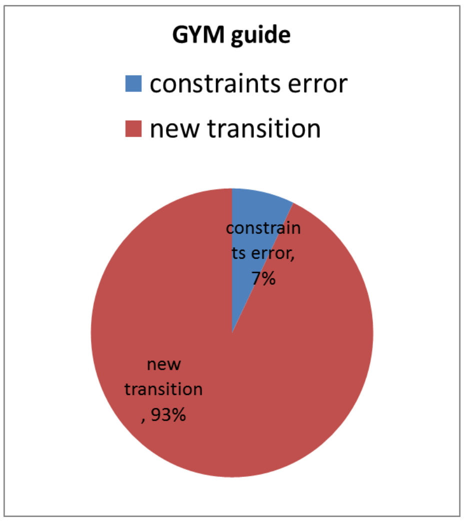


Figure 5-6 Experiment 2 error analysis (SUT2)

After analysis the report of the application “tippy tipper”, we found most of the error predicts occurred at meet a new transition verify. And the others errors are constraint errors.

Constraint errors are come from the test oracle is not well-trained. the invariant constraints hold the properties if where are none contradicted in the training data. If there is no counterexample of the fail constraint, invariant detector will set it true as invariant. With more and more training data, test oracle will discard the error constraints, and the remained constraints will be more reliable. The constraint error will lower with the increasing number of traces as training data.

New transition errors result from the transition we cannot verify. It might be a pass or a fail transition. If it is a fail transition, it should not happen in training data. By this way, an error transition is always a new transition for our Test Oracle. In another way, if it is a pass trace, it come from the transition is not happen before. It cause we don’t have the constraints corresponding to this transition. We can’t identify these two types, and it makes error prediction happened.

In the SUT “tippy tipper”, we found that all of the new transition errors come from one type of action, the “TEXT” action. While the screenshot has its block which is editable, the trace would try to edit the editable area. The departure of editing action is the “TEXT” type. 80%+ the wrong predict are because of it.

For GYM guide application. Most of the errors happen because of the type of the action “Rolling”, and we called it “scrolling” also. The argument of scrolling is -20 to +20. Then it will generate a lot of new action with low repetition, and those actions coverage is very low. However those actions are necessary for testing the applications completely, so we need a solution for this problem.

Both of tippy tipper and GYM guide applications, the true problem is the state explosion. It is a serious problem for our purpose. Several possible solutions for solving this problem, one solution is to redesign the action record, increasing the coverage of those actions, but it not an easy task. Another solution to solve the problem is ignored the actions with those types. In our experiment 2 environment, if we ignored those type of action, the hit rate of the tippy tipper application will reach about 95% hit rate, and the hit rate of the GYM guide application will reach 70%+ hit rate. It is a great increasing for our experiment results.

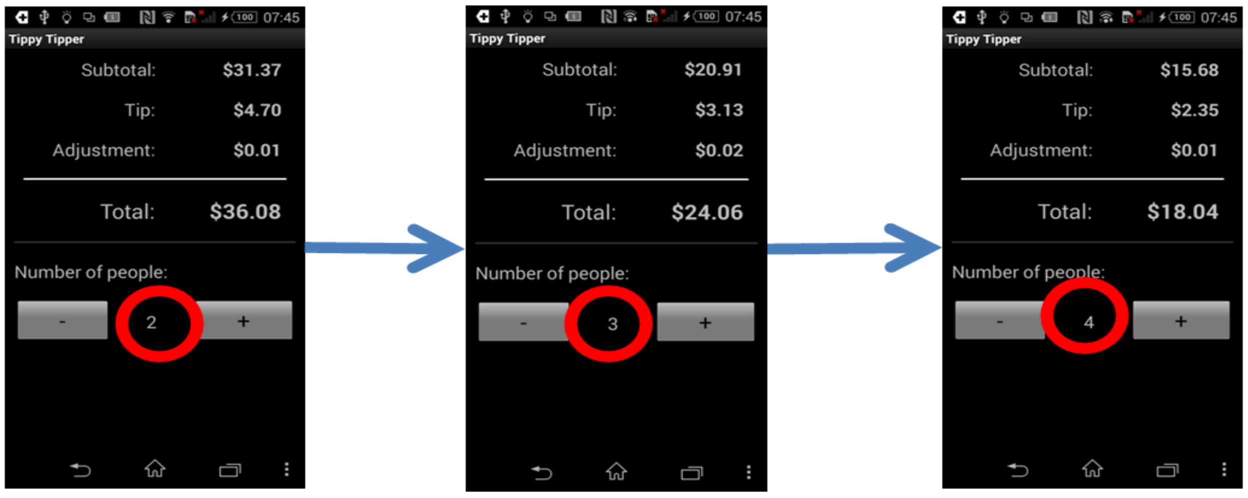
However there a risk come from the ignored strategy, if we ignore those actions, it means that we do not verify those type of actions, if one day, an error happen over those action, we cannot notice it happen. It is a big risk for us.

5.5 Meaningful invariant example

There is some meaningful invariant we got in the experiment

Example 1:

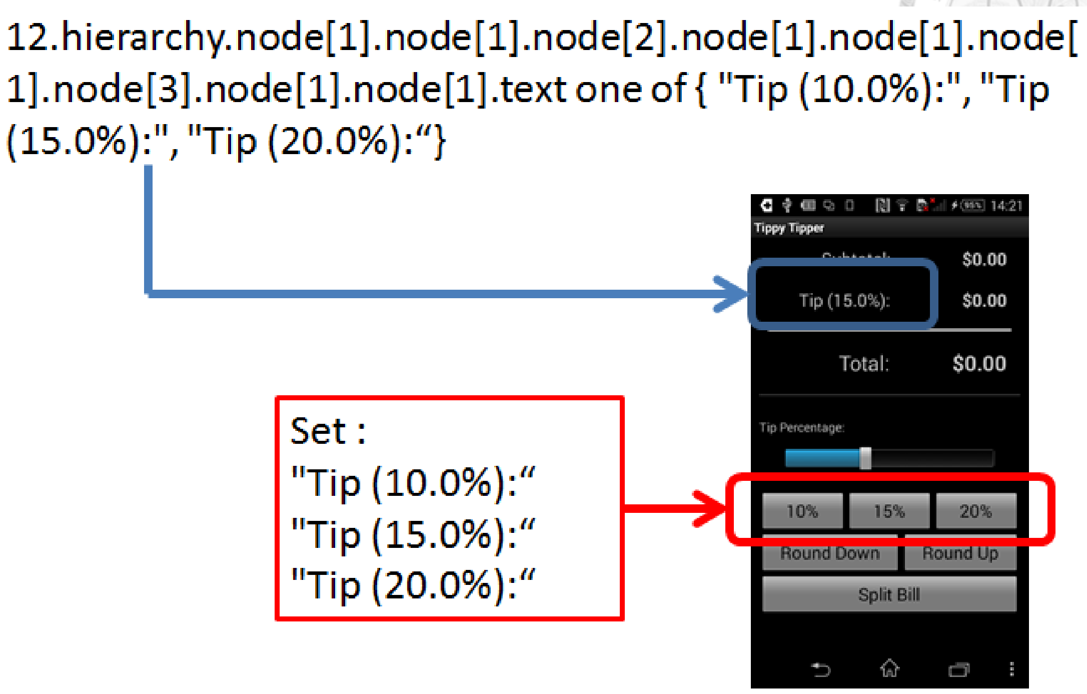
screen.15():::EXIT :14.hierarchy.node[1].node[1].node[2].node[1].node[1].node[1].node[9].node[1].node[1].node[2].text-orig(14.hierarchy.node[1].node[1].node[2].node[1].node[1].node[1].node[9].node[1].node[1].node[2].text) - 1 == 0


07:45
Tippy Tipper
Subtotal: $31.37
Tip: $4.70
Adjustment: $0.01
Total: $36.08
Number of people:
- 2 +

07:45
Tippy Tipper
Subtotal: $20.91
Tip: $3.13
Adjustment: $0.02
Total: $24.06
Number of people:
- 3 +

07:45
Tippy Tipper
Subtotal: $15.68
Tip: $2.35
Adjustment: $0.01
Total: $18.04
Number of people:
- 4 +

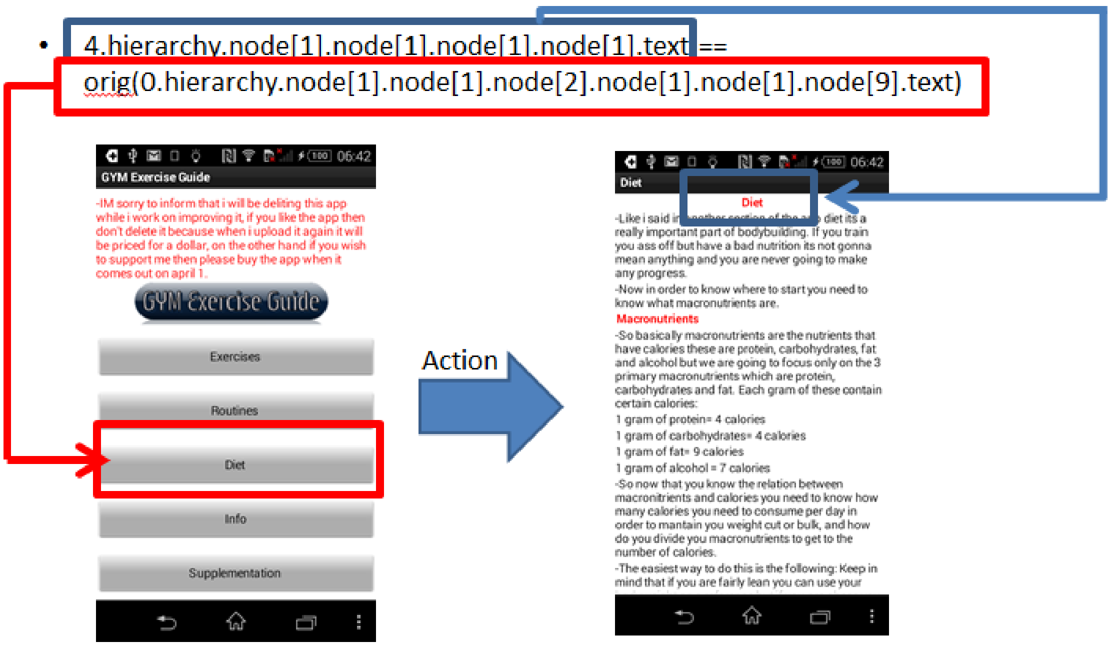
Figure 5-7 Meaningful invariant example(1)



12.hierarchy.node[1].node[1].node[2].node[1].node[1].node[1].node[3].node[1].node[1].text one of { "Tip (10.0%):", "Tip (15.0%):", "Tip (20.0%):"}

Set :
"Tip (10.0%):"
"Tip (15.0%):"
"Tip (20.0%):"

Figure 5-8 Meaningful invariant example(2)



4.hierarchy.node[1].node[1].node[1].node[1].text == orig(0.hierarchy.node[1].node[1].node[2].node[1].node[1].node[9].text)

-IM sorry to inform that i will be delting this app while i work on improving it, if you like the app then don't delete it because when i upload it again it will be priced for a dollar, on the other hand if you wish to support me then please buy the app when it comes out on april 1.

-Like i said in another section of the app diet its a really important part of bodybuilding. If you train you ass off but have a bad nutrition its not gonna mean anything and you are never going to make any progress.

-Now in order to know where to start you need to know what macronutrients are.

Macronutrients

-So basically macronutrients are the nutrients that have calories these are protein, carbohydrates, fat and alcohol but we are going to focus only on the 3 primary macronutrients which are protein, carbohydrates and fat. Each gram of these contain certain calories:

1 gram of protein= 4 calories

1 gram of carbohydrates= 4 calories

1 gram of fat= 9 calories

1 gram of alcohol = 7 calories

-So now that you know the relation between macronitrients and calories you need to know how many calories you need to consume per day in order to mantain you weight cut or bulk, and how do you divide you macronutrients to get to the number of calories.

-The easiest way to do this is the following: Keep in mind that if you are fairly lean you can use your

Figure 5-9 Meaningful invariant example(3)

Chapter 6 Conclusion

In this paper, we have purposed a reasonable system for constructing the test oracle of Android applications. The test oracle finds out the rules of the transitions of each Android application. The accuracy of the test oracle will increase as the size of the training set larger. There are still some problem needed to be resolved, such as state explosion and algorithm complexity problem. Ignoring the trouble actions is an easy solution, but it will also cause another problem. In order to handle more traces as training data, the algorithm complexity is becoming a serious problem of invariant detecting.

Chapter 7   Future Work

The state explosion is big problem to resolving. We need a better solution than what we did, and ignore the trouble actions might be one possible solution. The better solution should be a smarter strategy to abstract those actions.

We can reduce the complexity of the invariant detection algorithm by made our invariant detection tool up. Daikon is a well-done invariant detector for programs. However, there are many properties and grammar that we don’t need to check in xml transition testing. In order to fit the “dtrace” format, we modify the transition attributes and enlarge the number of elements for pass the Daikon check. If we make an invariant detector for XML, we can reduce the algorithm complexity by reduce the grammar types and reduce the variable number efficiently. It will be a better way to solve the complexity problem.

Reference

- "Using declarative specification to improve the understanding, extensibility, and comparison of model-inference algorithms"by Ivan Beschastnikh, Yuriy Brun, Jenny Abrahamson, Michael D. Ernst, and Arvind Krishnamurthy.
  IEEE Transactions on Software Engineering (2015)
- "Mining temporal invariants from partially ordered logs"
  by Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst, Arvind Krishnamurthy, and Thomas E. Anderson.
  SIGOPS Operating Systems Review (2011)
- "Building and using pluggable type-checkers"by Werner Dietl, Stephanie Dietzel, Michael D. Ernst, Kıvanç Muşlu, and Todd Schiller.
  In *ICSE'11, Proceedings of the 33rd International Conference on Software Engineering*,(2011)
- "Performance Enhancements for a Dynamic Invariant Detector"by Chen Xiao.
  Masters thesis, MIT Department of Electrical Engineering and Computer Science, (2007)
- "IODINE: A tool to automatically infer dynamic invariants for hardware designs" In the Design Automation Conference, DAC 2005.
- The Daikon system for dynamic detection of likely invariants Michael D. Ernst∗ , Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, Chen Xiao (2007)
- "Efficient Incremental Algorithms for Dynamic Detection of Likely Invariants" Jeff H. Perkins Michael D. Ernst MIT(2004)

- "Dynamically discovering likely program invariants to support program evolution"

  by Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin.

  IEEE Transactions on Software Engineering, vol. 27, no. 2, Feb. 2001
- "Dynamically Discovering Likely Program Invariants"

  by Michael D. Ernst.

  Ph.D. dissertation, University of Washington Department of Computer Science and Engineering, (Seattle, Washington), Aug. 2000
- "JDiff: A differencing technique and tool for object-oriented programs"

  by Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold.

  Automated Software Engineering Journal, vol. 14, Mar. 2007