

Automatic App Testing of LTL Properties *

Farn Wang^{*†} Ping-Hsuan Chen^{*} Wen-Hong Chiang^{*}

Dept. of Electrical Engineering^{*} & Graduate Institute of Electronic Engineering[†]

National Taiwan University

farn@cc.ee.ntu.edu.tw; <http://cc.ee.ntu.edu.tw/~farn>

Abstract

We present an automatic testing tool for Android apps. The tool accepts specifications in an extension of LTL (Linear Temporal Logic) and then automatically crawls the screen activities. We report experiments with the random test plan.

Keywords: software testing, mobile computing, app, LTL, crawling, Android

1 Introduction

Most *mobile computing application programs* (apps) are developed with stringent and fast evolution cycles. A popular app that has been downloaded tens of thousands times can be obsolete after several months in the market. Moreover, many such apps are developed by small teams that could not afford expensive development process.

In this work, we propose a tool, *Intelligent Test Oracle Lab (InTOL)*, for the black-box testing of Android apps. InTOL automatically crawls the screen activities of a given *software under test (SUT)*, records the event traces, and then drives test input to the Android device. Via a flexible application program interface, test engineers may implement their testing strategies and experiment with various verification techniques.

For test evaluation, we propose to use *linear-time temporal logic (LTL)* [3] as our specification language on screen activity traces. Many specification notations, including sequence diagrams, can be translated to LTL

quite straightforwardly. However, for the convenience in writing practical specifications, we propose the following adaptation.

- We extended LTL with working variables to allow us to save attribute values in a screen activity in such variables and later compare such variables with attribute values in another screen activity. For example, we may write

$$\Box\{x := \text{b1.clickable};\}(\text{b2.txt} = \text{"OK"} \rightarrow \bigcirc x = \text{b1.clickable})$$

to check whether in a screen activity with button ‘OK’, clickability of button ‘b1’ will remain the same in the next event. For convenience, we call the extension *extended LTL (ELTL)*.

- LTL properties are interpreted over infinite traces. In contrast, test evaluation is always carried out on finite traces. Havelund and Rosu [2] proposed LTL semantics on finite execution traces. However, they did not handle the need for evaluating the strength of finite test traces as an evidence for violation of LTL properties.

We propose to interpret LTL properties over finite traces with a prescribed repetition. For example, a property of the form $\Box(\eta_1 \rightarrow \Diamond\eta_2)$ may specify that when η_1 is true at a screen activity, eventually η_2 will become true in a future screen activity. An infinite trace that violates the property must contain a screen activity a satisfying η_1 such that all screen activities no earlier than a in the trace must not satisfy η_2 . In practice, it is not possible to test an infinite trace.

In evaluating such a property, we allow the users to prescribe a repetition count k . Specifically, InTOL reports violation of property $\Box(\eta_1 \rightarrow \Diamond\eta_2)$ by a finite trace when in the trace, there is a

*The work is partially supported by NSC, Taiwan, ROC under grants NSC 95-2221-E-002-067 and NSC 95-2221-E-002-072. A full version of the paper with supplemented proofs can be found in <http://cc.ee.ntu.edu.tw/~farn/publications.htm>.

screen activity a satisfying η_1 such that following a , some screen activity repeats $k + 1$ times. The evaluation shows the following two facts.

- The repetition pattern following activity a can be extended to a hypothetical infinite trace that violates the property.
- The trace exhibits enough number of repetitions, as prescribed by parameter k , after activity a to predict the eventual violation.

We have experimented with the random test plans. We have also experimented with three Android apps from Google play. Preliminary result shows that InTOL is effective in revealing bugs.

2 Architecture of InTOL

Our tool, InTOL, runs on a PC and tests an Android device connected to the PC via a USB line. InTOL consists of the following modules.

- **Screen activity extractor (SAE)** is responsible for extracting the screen activities of the Android device via UIAutomator and saves the screen activities in XML format to our database.
- **Test case generator (TCG)** is a user-supplied module complying to the program interface to InTOL. TCG can query the DB of InTOL and issue test directives on actionable views in the screens through the program interface. TCG may also send directives to start and end a test case.
- **Test input adaptor (TIA)**. Upon receiving the test directives, InTOL first records the test directives to our DB. The test directives and the extracted screen activities together are interleaved and recorded as our event traces. TIA then translates the test directives to pixel-level test input and drives the test input to the device via adb.
- **Test trace evaluator (TTE)**. After finishing executing a test case, TTE checks whether the just-finished event trace is an evidence for the violation of the given LTL properties. If it is, then a test report is issued. When the SUT stops execution, InTOL also extracts the system calling stack, also in an XML format, and appends the stack contents to the end of the execution trace. Thus, InTOL can also tell whether the SUT crashed.

3 Test evaluation

Linear-time temporal logic (LTL) was proposed by Pnueli as a formal language for specifying and reasoning about parallel programs [3]. It is well-studied and equipped with many algorithms for model analysis. In this work, we extend LTL with working variables and inequalities for specifications on attribute values in execution traces.

3.1 Event traces

Assume that A is the set of attribute names. A valuation of A is a mapping from A to values. An event trace θ is a finite sequence of valuations of A . We use $|\theta|$ to denote the number of elements of θ . Conceptually, θ is a mapping from $[0, |\theta| - 1]$ to valuations of A . In this work, an event trace is an alternating sequence of screen activities and actions.

3.2 Extended Linear-time Temporal Logic (ELTL)

ELTL formulae (ranged over by $\phi, \psi, \dots$) are of the syntax in table 1. Here “ $\sim$ ” is an inequality operator, “ c ” is a natural number, “ a ” is an attribute in A , “ x ” is a working variable declared in the LTL property, and $+$, $-$, $\cdot$, $/$, $\%$ are respectively the addition, subtraction, multiplication, division, and the remainder operators.

For example, “*buttonReset.clickable=0*” may specify that the reset button is not *clickable* in the current screen. For a Boolean attribute a , we may use a as a shorthand for $a = 1$ and $\neg a$ for $a = 0$.

Operators $\neg$ and $\vee$ are Boolean negation and disjunction respectively. $\bigcirc$ and $\mathbf{U}$ are temporal operators ‘next’ and ‘until’ respectively. Boolean connectives such as $\wedge$, $\rightarrow$, and $\leftrightarrow$ are derived operators from $\neg$ and $\vee$. Other useful shorthands include $false \equiv \neg true$, $\Diamond \phi \equiv true \mathbf{U} \phi$ (the eventually operator), and $\Box \phi \equiv \neg \Diamond \neg \phi$ (the always operator). In literature, $\bigcirc$, $\Diamond$, and $\Box$ are also written as “X”, “F”, and “G” respectively [1].

The satisfaction of an ELTL formula ϕ by an infinite event trace θ , written $\theta \models \phi$, is similarly defined as that of an LTL formula. Due to page limit, we leave the satisfaction relation, $\theta \models \phi$, in appendix A.

$$\begin{aligned}
\phi &::= \text{true} \mid \epsilon \sim c \mid \neg\phi \mid \phi \vee \psi \mid \{x := \epsilon;\} \phi \mid \bigcirc \phi \mid \phi \mathbf{U} \psi \\
\epsilon &::= c \mid a \mid x \mid \epsilon_1 + \epsilon_2 \mid \epsilon_1 - \epsilon_2 \mid \epsilon_1 \cdot \epsilon_2 \mid \epsilon_1 / \epsilon_2 \mid \epsilon_1 \% \epsilon_2
\end{aligned}$$

Table 1. Syntax of ELTL

3.3 Evaluation of LTL on finite traces

We adopt the following approach. Given two finite traces τ and ρ with $\tau(|\tau| - 1) = \rho(0)$ and $\rho(0) = \rho(|\rho| - 1)$, we intuitively use $\tau(\rho)^\omega$ to denote an infinite trace constructed by executing τ first and then repeating ρ infinitely many times. (Each time ρ is executed, we skip its first event.)

Given a finite event trace θ , an ELTL property ϕ , and an integer constant k , we say that θ violates ϕ with strength k , in symbols $\theta \not\models_{(k)} \phi$, if and only if θ is the concatenation of an initial segment τ and a finite tail ρ with the following constraints.

- ρ is a cycle of screen activities and actions.
- τ is a segment from the initial screen activity to ρ .
- $\tau(\rho)^\omega$ is an evidence of violation of ϕ .
- $\rho(0)$ appears $k + 1$ times in ρ .

Intuitively, the truth of $\theta \not\models_{(k)} \phi$ shows that in θ , we have waited for k identical screen activities and still could not find an evidence for the satisfaction of property ϕ .

At the moment, InTOL focuses on the evaluation of three types of app properties. For each type, we give an example and explain how we evaluate the violation by traces.

3.4 State safety properties

A state safety property ϕ is of the form $\Box\eta$ where η is a Boolean constraint on attribute values. Such a property specifies the constraint that must be satisfied at every screen activity and action. When InTOL evaluates such a state safety property over a finite trace θ , it sets $k = 1$. Specifically, $\theta \not\models_{(1)} \phi$ means the following.

- η is violated at least once in θ .
- θ can be extended to an infinite event trace as a witness for the violation of ϕ .

3.5 Transition safety properties

Such a property is of the form $\phi : \Box(\eta_1 \rightarrow \bigcirc(\eta_2 \rightarrow \bigcirc\eta_3))$. The property can be used to specify the behavior between screen activities in actions. Specifically, η_1 is a constraint on the screen activity before the action, η_2 is on the action, and η_3 is on the activity after the action. Thus the property says that in every screen activity satisfying η_1 , when an action satisfying η_2 is taken, a screen activity satisfying constraint η_3 will follow.

When InTOL evaluates such a transition safety property over a finite trace θ , it also sets $k = 1$. Specifically, the truth of $\theta \not\models_{(1)} \phi$ means the following.

- At least once in θ , after an action satisfying η_2 in a screen activity satisfying η_1 , we do not see a screen activity satisfying η_3 .
- θ can be extended to an infinite event trace as a witness for the violation of ϕ .

3.6 Responsiveness properties

Such a property is of the form: $\phi : \Box(\eta_1 \rightarrow \Box(\eta_2 \rightarrow \Diamond\eta_3))$ and can be used to specify the expected response in an indefinite period. For instance, η_1 is a constraint on the screen activity before the action, η_2 is on the action, and η_3 is on the activity that should eventually happen. Thus the property says that after events satisfying respectively η_1 and η_2 , a screen activity satisfying η_3 will eventually follow. For such a property, InTOL allows the users to prescribe a suitable value of k . Thus the truth of $\theta \not\models_{(k)} \phi$ means the following.

- At least once in θ , an event satisfying η_1 followed by another event satisfying η_2 has been observed.
- In θ , we have waited long enough, in fact $k + 1$ repetitions of certain events, for an evidence of η_3 and still could not observe it.
- θ can be extended to an infinite event trace as a witness for the violation of ϕ .

4 Implementation

We have implemented InTOL for the black-box testing of Android Apps. No instrumentation to Android and the SUTs is required. InTOL server adopts a LAMP framework (Linux, Apache, Microsoft, and Perl) and runs on VMWare Player 4.0.3 which in turn runs on the Linux environment (Ubuntu 10.04) with two GB RAM and 1 processor core.

Once the app under test (i.e., SUT) is started in the device, InTOL then repetitively extracts the screen activities of the Android device via UIAutomator, records the screen activities (in XML format), and drives test input to the device via adb. When the SUT stops execution, InTOL also extracts the system calling stack, also in an XML format, and appends the stack contents to the end of the execution trace. Thus, InTOL can also tell whether the SUT crashes.

5 Experiment report

We set up the following experiments with the following apps in the Google Play as benchmarks.

- *Dalvik Explorer* (*org.jessies.dalvikexplorer*). We also put down eight LTL properties as our specifications.
- *E Numbers* (*org.uaraven.e*). We put down seven LTL properties as its specification.
- *Mathdroid* (*com.yt.org.jessies.mathdroid*). We put down 5 LTL properties as its specification.

The source codes in Java of the three benchmarks are available publicly. We randomly inserted bugs to the source codes of the three benchmarks. Bug density is set to 1 bugs per 1000 lines of source code.

We use random testing as our test case generation strategies. Specifically, we randomly generate test cases of 200 to 500 screen activities and action. We stop only when all inserted bugs have been revealed.

In our experiment, all the inserted bugs were revealed by InTOL. Table 2 shows that InTOL is capable of revealing all the inserted bugs. In the experiment, the users only have to put down the specifications in ELTL and all the black-box testing sessions are then automatic. In general, writing specifications can be in a much higher-level of abstraction than writing test cases.

SUT	number of traces	number of total actions
<i>Dalvik Explorer</i>	981	48970
<i>E Numbers</i>	94	23500
<i>Mathdroid</i>	1165	36175

Table 2. Experiment data

6 Conclusion

We present a black-box testing tool, InTOL, for Android apps for specifications as extended LTL properties. Users only have to write down their specifications and all the test sessions can then be automatically executed.

InTOL also has a flexible application program interface that allows the users to plug in their own testing strategies. In the future, it would be interesting to see how different testing strategies can be experimented with InTOL.

References

- [1] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Workshop on Logic of Programs*, volume LNCS 131. Springer-Verlag, 1981.
- [2] K. Havelund and G. Rosu. Testing linear temporal logic formulae on nite execution traces. Technical report, NASA, 2001.
- [3] A. Pnueli. The temporal logic of programs. In *18th annual IEEE-CS Symposium on Foundations of Computer Science*, pages 45–57, 1977.

Appendix

A Semantics of ELTL

An environment on a variable domain X is a mapping from X to values of variables in X . An ELTL property is interpreted with an environment $\mathcal{E}$ and a finite trace θ . We let $\llbracket \theta \rrbracket$ be the environment that maps all attributes in the first screen activity (i.e., $\theta(0)$) of θ to their corresponding values in $\theta(0)$.

We also use $[a := c;]$ to denote the partial environment that interprets a as c . Given two partial environments $\mathcal{E}_\infty$ and $\mathcal{E}_\epsilon$, we use $\mathcal{E}_\infty \mathcal{E}_\epsilon$ to denote the valuation that interprets an attribute a as in E_2 if it is defined in E_2 ; and as in E_1 otherwise.

Given an ELTL formula ϕ , an infinite trace θ , and an environment $\mathcal{E}$, θ satisfies ϕ in environment $\mathcal{E}$, in symbols $\theta \models_{\mathcal{E}} \phi$, if and only if the following inductive relations are all satisfied.

- $\theta \models_{\mathcal{E}} \text{true}$
- $\theta \models_{\mathcal{E}} \epsilon \sim c$ if and only if $\mathcal{E} \llbracket \theta(0) \rrbracket (\epsilon) \sim c$, i.e., in the composed environment of $\mathcal{E}$ and $\llbracket \theta(0) \rrbracket$, $\epsilon \sim c$.
- $\theta \models_{\mathcal{E}} \neg \phi$ if and only if $\theta \models \phi$ is false.
- $\theta \models_{\mathcal{E}} \phi \vee \psi$ if and only if $\theta \models_{\mathcal{E}} \phi$ or $\theta \models_{\mathcal{E}} \psi$.
- $\theta \models_{\mathcal{E}} \{x := \epsilon; \} \phi$ if and only if $\theta \models_{\mathcal{E}[x:=\mathcal{E}(\epsilon)]} \phi$.
- $\theta \models_{\mathcal{E}} \bigcirc \phi$ if and only if $\theta[1 \dots] \models_{\mathcal{E}} \phi$. Here $\theta[k \dots]$ is $\theta(k)\theta(k+1) \dots$.
- $\theta \models_{\mathcal{E}} \phi \mathbf{U} \psi$ if and only if there exists an $k \in \mathbb{N}$ with the following restrictions.
 - $\theta[k \dots] \models_{\mathcal{E}} \psi$.
 - For any $j \in [0, k-1]$, $\theta[j \dots] \models_{\mathcal{E}} \phi$.

Intuitively, $\phi \mathbf{U}_{\mathcal{E}} \psi$ means that ψ is true either now or in the future and ϕ holds since now until that moment.