Reinforcement Learning-Driven Test Generation for Android GUI Applications using Formal Specifications

Yavuz Koroglu
Ph.D. Candidate

E-Mail: yavuz.koroglu@boun.edu.tr
Web: https://www.cmpe.boun.edu.tr/~yavuz.koroglu/
Department of Computer Engineering
Bogazici University, Istanbul/Turkey

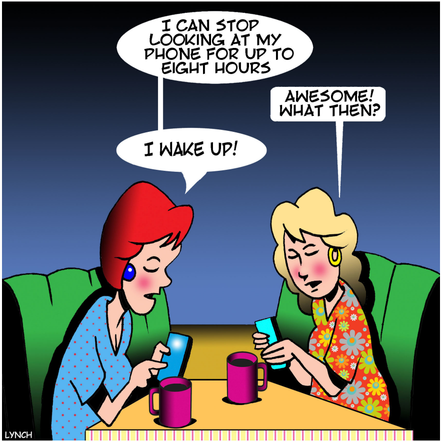
December 3, 2019

Contents

Motivation of Testing

- **Inadequate Testing** may have a very **high cost**.
  - Knight Capital Group's $440M bug.
  - Pentium FDIV Bug - $475M.
  - Morris Worm - $100K - $10M.
- **Adequate Testing** requires **time + effort**.
- Formal Methods:
  - \+ Complete
  - \- Not scalable
- Testing:
  - \- Incomplete tests
  - \+ Scalable
  - Still, in Microsoft, 79% of the developers are dedicated to writing unit tests.
- **Automated Test Generation**
  - **Decreases time and effort** of testing while makes the approach more complete.
  - **Large** body of work, but
  - **Limited** real-world usage.

Motivation of Android



CartoonStock.com

Mobile GUI Applications are Ubiquitous

- We use mobile phones often **(3 hours/day)**
- Mostly on mobile applications **(90% of the time spent)**

Android Market is Growing

- **2.6 billion** mobile phone users

Android has the Largest Share

- **82.8%** of all apps are for Android

Problems in Mobile Market

App Fatigue

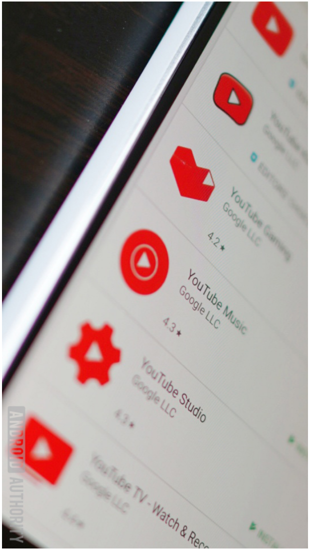
Too many apps for similar tasks.

Incomplete Apps

Some apps **fail** to perform their **intended tasks**.

Fake Apps

Some apps are completely **fake**.



Recent Study on Fake Apps @ ICSE'19

2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

A Large-Scale Empirical Study on Industrial Fake Apps

Chongbin Tang*, Sen Chen*, Lingling Fan*, Lihua Xu†, Yang Liu‡, Zhushou Tang§, Liang Dou*
*East China Normal University, China †New York University Shanghai, China
‡Nanyang Technological University, Singapore §Pwnzen Infotech Inc., China

***Abstract*—While there have been various studies towards Android apps and their development, there is limited discussion of the broader class of apps that fall in the fake area. Fake apps and their development are distinct from official apps and belong to the mobile underground industry. Due to the lack of knowledge of the mobile underground industry, fake apps, their ecosystem and nature still remain in mystery.**

To fill the blank, we conduct the first systematic and comprehensive empirical study on a large-scale set of fake apps. Over 150,000 samples related to the top 50 popular apps are collected for extensive measurement. In this paper, we present discoveries from three different perspectives, namely fake sample characteristics, quantitative study on fake samples and fake au-

of app searching and downloading is greatly affected by the fake apps in real world.

Even worse, as the doorsill to develop an app has been set low, the cost to develop a fake app is much lower than what it takes to develop a desktop program, providing an ideal hotbed for the underground industry to thrive on [3]. Moreover, the flexibility of Android app implementation [4] contributes the fake apps' complexity.

Despite the ubiquity, little is known about fake apps and their ecosystem – their common characteristics, the number of fake apps at large, their production process and speed,

Solution: Testing

Functional Testing will reveal many incomplete and fake apps.

Test Automation

Currently, **test automation** tools (e.g. Appium) are common for functional testing.

- ✔ **Helps Developers :** To design functional tests.
- ✖ **Requires Manual Effort :** The developer must
  1. Generate (input data etc.),
  2. Execute (must observe execution), and
  3. Evaluate (check if the output agrees with expectations)

  tests, all manually.

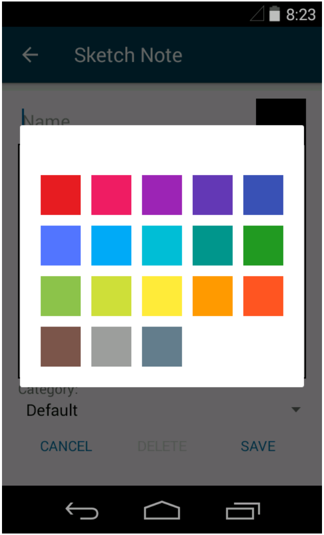
Motivation for Android GUI Testing

Current Situation

- Millions of apps
  - Pressure on developers to **continuously develop**,
  - Need for **functional testing**.
- Fragmentation
  - Many **OS versions**, many **devices**.
  - **Portability** issues.
- Fake apps and unimplemented functions
  - Does the app implement its **promised function**?
- Bug reports and customer feedback
  - Developer needs to **verify**.
- An **automated test generation tool** would
  - Ease the burden on the developer.

Scenario #1: Verifying Bugs

- A Notes application.
  - Allows **drawing sketches**.
- A user **reports an issue**.
  - Black is missing from the color palette.
- Developer **has to find**
  - The buggy screen.



Scenarios of Android GUI Testing II

Scenario #2: Functional Testing

- Developer **recently added a function**.
  - Playing against AI in a chess game.
- Developer **has to verify** that
  - The AI indeed makes a move.

Scenarios of Android GUI Testing III

Scenario #3: Robustness Against Fragmentation

- Developer **has to verify** that the chess AI works on
  - Different platforms (OS) and
  - Different devices.

Scenario #4: Non-functional Testing

- Developer **has to verify** that
  - The chess AI makes a move in less than 3 seconds.

Related Work in Android Testing

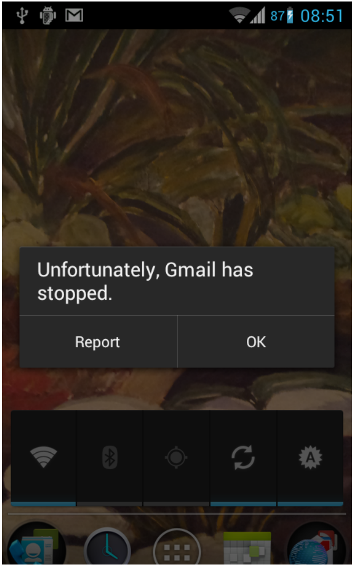
Test Generation Engines for Android (Alphabetically Ordered)

1. ✔ A³E
2. ✔ ACTEve[4]
3. ✔ CrashScope
4. ✔ CrawlDroid
5. ✔ DroidBot
6. ✔ DynoDroid[3]
7. ✖ EvoDroid
8. ✖ LAND
9. ✖ MATE
10. ✔ MobiGUITAR
11. ✔ Monkey (Google)
12. ✖ MonkeyLAB[2]
13. ✖ ORBIT[4]
14. ✔ PUMA
15. ✖ QBE[1]
16. ✖ QUANTUM
17. ✖ Sapienz (Facebook)
18. ✔ Stoat
19. ✔ SwiftHand (UCB)
20. ✔ SwiftHand2
21. ✖ TCM[1,2]

✔ publicly available (12) ✖ unavailable (9)

1 Our previous work.
2 Requires an initial set of test cases.
3 Instruments Android OS.
4 Requires the source code.

Problems of Existing Test Generation Engines



Focus on Fatal Exceptions Only

- ✔ **Very simple** test oracle.
- ✖ **Ignore** other bugs.

Focus on Structural Coverage

- Code, method, activity etc.
- ✖ NOT **functional**.
  - Tests may cover many activities but
  - Fail to test **essential functions**.

Example

Start a chess game but do NOT **move**.

Test Oracles

Definition

Says a test has **passed** or **failed**.

Implicit Test Oracle

- ✔ Automated.
- ✖ Implemented.
- ✖ Not scalable.

Test Oracles

Definition

Says a test has **passed** or **failed**.

Implicit Test Oracle

- ✔ Automated.
- ✖ Implemented.
- ✖ Not scalable.

Example

- Fatal Exceptions.
- Activity Coverage.

Test Oracles

Definition

Says a test has **passed** or **failed**.

Implicit Test Oracle

- ✔ Automated.
- ✖ Implemented.
- ✖ Not scalable.

Specified Test Oracle

- ✖ Developer writes specs.
- ✔ Monitorable.
- ✔ Scalable.

Example

- Fatal Exceptions.
- Activity Coverage.

Test Oracles

Definition

Says a test has **passed** or **failed**.

Implicit Test Oracle

- ✔ Automated.
- ✖ Implemented.
- ✖ Not scalable.

Specified Test Oracle

- ✖ Developer writes specs.
- ✔ Monitorable.
- ✔ Scalable.

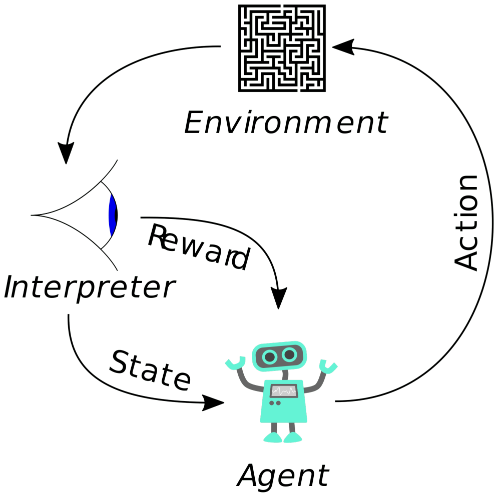
Example

- Fatal Exceptions.
- Activity Coverage.

Moreover, a specified test oracle

- is a **formal specification**.
- ✔ Unambiguous.

Motivation for Reinforcement Learning (RL)



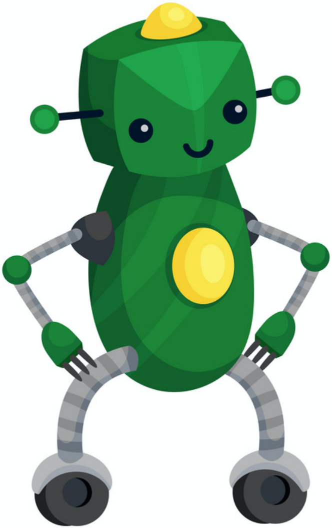
High performance in

- ✔ Resource management,
- ✔ Traffic control,
- ✔ Chess,
- ✔ Atari...

Also,

- ✔ Requires **no labeled data** (unlike ANN).
- ✔ Learns from **trial-and-error**.
- ? Requires an **interpreter** to generate rewards.

RL for Test Generation



Typically,

Trained until convergence,

- Learns to perform a task **indefinitely**.

Example

Standing robot (on the left)

Testing

- Generate **once and terminate**.
- Do NOT wait for convergence.

Summary

- Need automated test generation for
  - **functional testing** and
  - **bug verification**.
- Existing automated test generation engines are **inadequate**.
- Introduced
  - **specified test oracles** and
  - **reinforcement learning**.
- Emphasized
  - RL for testing $\Rightarrow$ Do NOT wait for **convergence**.

FARLEAD-Android Overview

Fully Automated Reinforcement LEArning-Driven Specification-Based Test Generator for Android

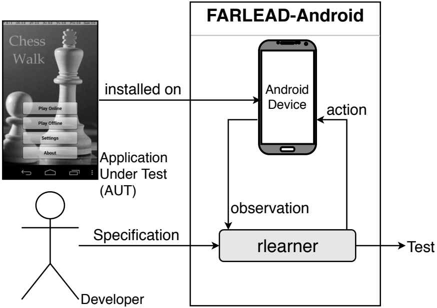


Figure: FARLEAD-Android Overview

Takes

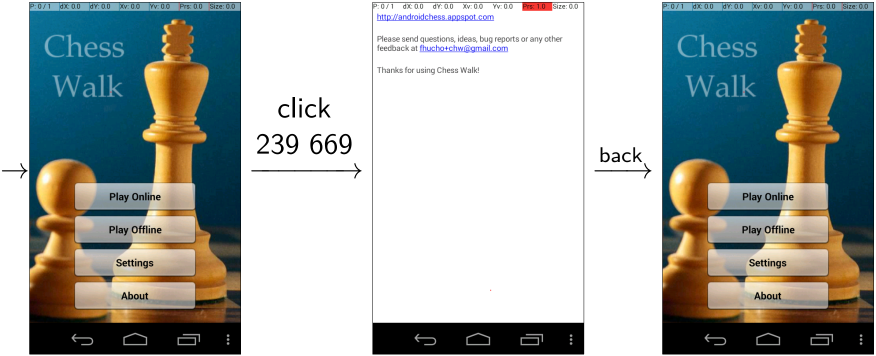
- The **app binary** (.apk) and
- A **specification** (spec)

Crawls the app

- Monitors the **spec**.
- Outputs a **witness**.
  - A replayable **test**.

Proceed to DEMO

Example Spec: Traversing Menus



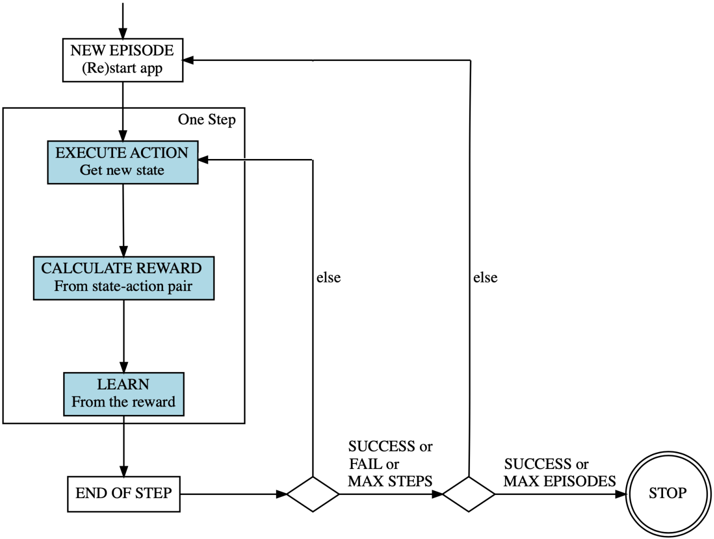
Example (Linear-time Temporal Logic, LTL, Spec)

$\varphi = \bigcirc ([\text{act.} \sim \text{Main}] \wedge ([\text{act.} \sim \text{Main}] \mathcal{U} ([\text{act.} \sim \text{About}] \wedge ([\text{act.} \sim \text{About}] \mathcal{U} [\text{act.} \sim \text{Main}]))))$

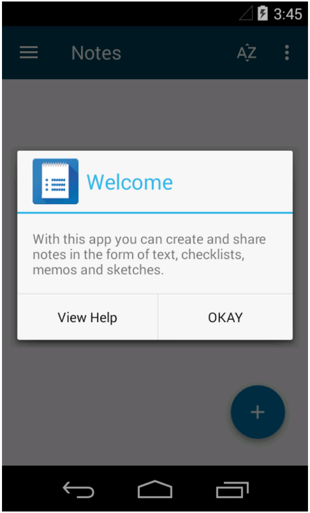
Description: Main activity is open in the next state, then Main activity is open until About activity is open, and then About activity is open until Main activity is open again.

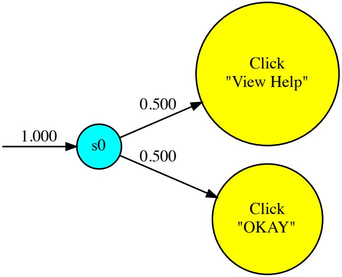
Proceed to DEMO

The RL Agent (rlearner)



What does rlearner learn? (Policy)

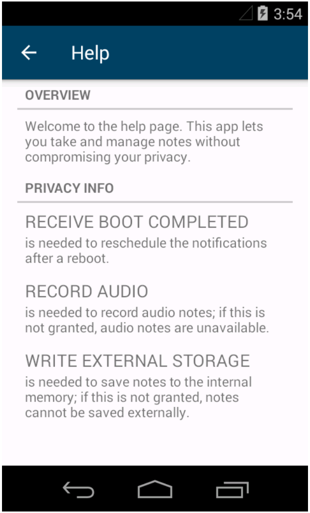


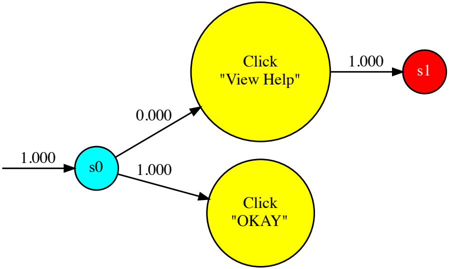


Policy

is a **markov chain**.

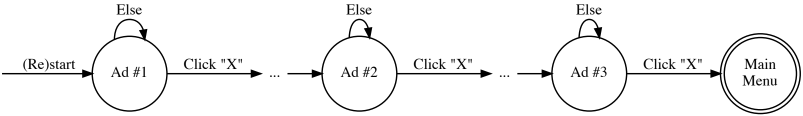
What does rlearner learn? (Policy)





Policy

is a **markov chain**.



Example Spec

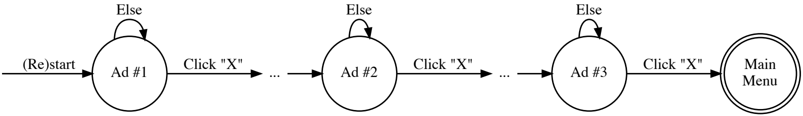
$$\varphi = \bigcirc([\text{Ad } \#1] \wedge \bigcirc([\text{Ad } \#2] \wedge \bigcirc([\text{Ad } \#3] \wedge \bigcirc[\text{Main}])))$$

Description: In the next state,

1. Ad #1, next,
2. Ad #2, next,
3. Ad #3, and finally,
4. Main

Problem:

- Ad #1 $\xrightarrow{\text{Click "X"}}$ gets reward.
- Click "X" did NOT get reward for **future states**.



Example Spec

$$\varphi = \bigcirc([\text{Ad } \#1] \wedge \bigcirc([\text{Ad } \#2] \wedge \bigcirc([\text{Ad } \#3] \wedge \bigcirc[\text{Main}])))$$

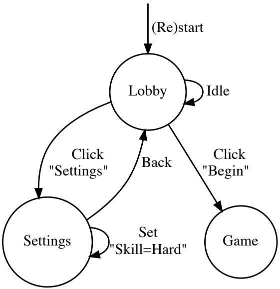
Description: In the next state,

1. Ad #1, next,
2. Ad #2, next,
3. Ad #3, and finally,
4. Main

Solution:

- Learn **stateless** action values.
- Give reward to $\xrightarrow{\text{Click "X"}}$.
- **Initialize** state-action values with action values.

Improvement #2: Tails/Decisions



Example Spec

$\varphi = \Diamond([\text{Skill} = \text{Hard}] \wedge \Diamond(\text{Lobby} \wedge \Diamond\text{Game}))$

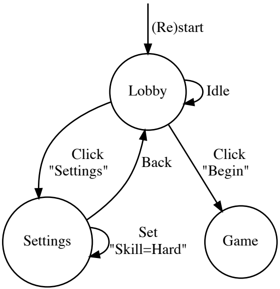
Description: Eventually,

1. Set Skill to Hard, then
2. Go to Lobby, then
3. Start the Game.

What If?

- The first test sets the Skill to Hard, then goes to Lobby, then Idle.
- Settings $\xrightarrow{\text{Back}}$ gets **high reward**.
- Further episodes get **STUCK** at Lobby→Settings→Lobby.

Improvement #2: Tails/Decisions



Example Spec

$\varphi = \Diamond([\text{Skill} = \text{Hard}] \wedge \Diamond(\text{Lobby} \wedge \Diamond\text{Game}))$

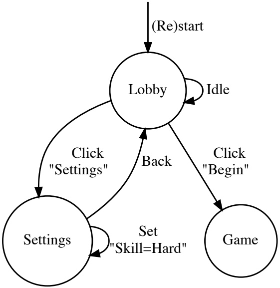
Description: Eventually,

1. Set Skill to Hard, then
2. Go to Lobby, then
3. Start the Game.

Solution: Tails/Decisions

- Rewards depend on **history**.
- Replace states S with **tails** $\mathcal{S} = \bigcup_{i=0}^{h} (A \times S)^h$.
- Replace actions A with **decisions** $\mathcal{A} = \mathcal{S} \times A$.

Improvement #2: Tails/Decisions



Example Spec

$\varphi = \Diamond([\text{Skill} = \text{Hard}] \wedge \Diamond(\text{Lobby} \wedge \Diamond \text{Game}))$

Description: Eventually,

1. Set Skill to Hard, then
2. Go to Lobby, then
3. Start the Game.

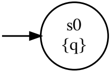
Example

- Do NOT give reward to Settings $\xrightarrow{\text{Back}}$.
- Give reward to Settings $\xrightarrow{\text{Skill=Hard}}$ Settings $\xrightarrow{\text{Back}}$.

How to Calculate Rewards?

Example

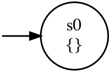
- $\varphi = p \, \mathcal{U} \, q$ (p must be true until q becomes true)



If $q = \top$,

✔ $p \, \mathcal{U} \, q = \top$.

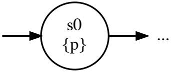
- Reward $\leftarrow 1$.



If $p = q = \bot$,

✖ $p \, \mathcal{U} \, q = \bot$.

- Reward $\leftarrow -1$.



If $p = \top$ ($q = \bot$),

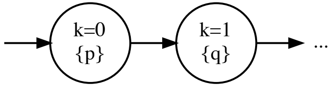
? $p \, \mathcal{U} \, q =?$.

- Reward $\leftarrow 0$.

Main Idea

At every step,

- New spec ($\varphi_{s_{k+1}}$) $\leftarrow$ **modify** the current spec (φ_{s_k}).
- Reward $\leftarrow$ some **distance metric**.



- $\varphi_{s_0} = p\,\mathcal{U}\,(q \wedge \bigcirc[q\,\mathcal{U}\,p])$
- $\varphi_{s_1} = q\,\mathcal{U}\,p$
- ? Reward $\leftarrow$ 0?
- ? **Intermediate** rewards?

Reward Shaping

$$r = \begin{cases} -1 & \varphi_{s_{k+1}} = \neg\top \\ \frac{|N(\varphi_{s_k}) - N(\varphi_{s_{k+1}})|}{N(\varphi_{s_k}) + N(\varphi_{s_{k+1}})} & \text{otherwise} \end{cases}$$

where φ_{s_k} is the spec in state s_k, N is the **reward metric** function that returns the **number of atomic propositions** in φ.

Example

- $\varphi_{s_0} = p\,\mathcal{U}\,(q \wedge \bigcirc[q\,\mathcal{U}\,p])$
- $\varphi_{s_1} = q\,\mathcal{U}\,p$
- $r \leftarrow \frac{|N(\varphi_{s_0}) - N(\varphi_{s_1})|}{N(\varphi_{s_0}) + N(\varphi_{s_1})} = \frac{|4-2|}{4+2} \approx .33$

Experimental Scenarios I

Experimental Setup

- Two Android GUI Applications (Notes and ChessWalk),
- Nine scenarios.

Scenario #1: ChessWalk - Function

- **Description:** The user goes to the AboutActivity and returns back.

Scenario #2: ChessWalk - Function

- **Description:** The user goes to the SettingsActivity and returns back.

Experimental Scenarios II

Scenario #3: ChessWalk - Function

- **Description:** Pausing and resuming the application should not change the screen.

Scenario #4: ChessWalk - Bug Report

- **Description:** The application should prevent the device from sleeping BUT it does NOT.

Scenario #5: ChessWalk - Function

- **Description:** Changed settings should be remembered later.

Experimental Scenarios III

Scenario #6: ChessWalk - Function

- **Description:** The user starts a game and make a move.

Scenario #7: ChessWalk - Bug Report

- **Description:** Second game shows the moves of the first game.

Scenario #8: Notes - Bug Report

- **Description:** Black is missing from a color palette.

Scenario #9: Notes - Bug Report

- **Description:** Even if a note is canceled, it is still created.

Scenarios ⇒ LTL (Levels of Detail)

Level (a) – Declarative

Only propositions about,

- ✔ states
- ✖ action types
- ✖ action details

Level (b) – Mixed

Only propositions about,

- ✔ states
- ✔ action types
- ✖ action details

Level (c) – Imperative

All propositions about,

- ✔ states
- ✔ action types
- ✔ action details

Note that, we expect

Test Generation

- **Slow** in level (a).
- **Fast** in level (c).

Scenarios $\Rightarrow$ LTL (Levels of Detail)

Example (Declarative - Level a)

$\bigcirc([\text{activity} \sim \text{Main}]\mathcal{U}([\text{activity} \sim \text{About}] \wedge \bigcirc([\text{activity} \sim \text{About}]\mathcal{U}[\text{activity} \sim \text{Main}]))$

Example (Mixed - Level b)

$\bigcirc(([\text{act.} \sim \text{Main}] \wedge \text{action} = \text{click})\mathcal{U}([\text{act.} \sim \text{About}] \wedge \bigcirc([\text{action} = \text{back}]\mathcal{U}[\text{act.} \sim \text{Main}]))$

Example (Imperative - Level c)

$\bigcirc((([\text{action} = \text{click}] \wedge [\text{actionDetail} \sim \text{About}]) \wedge [\text{act.} \sim \text{About}]) \wedge \bigcirc([\text{action} = \text{back}]\mathcal{U}\text{act.} \sim \text{Main}))$

Imperative LTL

- Write test cases in LTL.
- ✔ Do NOT write **Java/Kotlin**.

- ✔ **Portable**.
- ✔ **Maintainable**.

Experimental Setup

Engines Under Experimentation

1. **Random:** Random exploration.
2. **Monkey:** Random exploration with built-in monkey actions.
3. **QBEa:** Q-Learning Based Exploration optimized for activity coverage.
4. **FARLEADa:** FARLEAD-Android with Level (a) specs.
5. **FARLEADb:** FARLEAD-Android with Level (b) specs.
6. **FARLEADc:** FARLEAD-Android with Level (c) specs.

Re-implemented Other Engines in FARLEAD-Android

- Need to **monitor** LTL specs.

Experimental Setup

For

- Every engine (6 engines),
  - Every scenario (9 scenarios), and
    - Execute test generation 100 times, for
    - Maximum 500 episodes and
    - Maximum 4 or 6 steps (depending on scenario).
- Execute on VirtualBox guest with
  - Android 4.4 OS
  - 480x800 screen resolution
- Virtual machine is **advantageous** over a physical device

✔ Reproducibility. | ✔ Scalability. | ✔ Configurability.

- Measure
  - **Effectiveness** and
  - **Performance**.

Effectiveness Results

Engine \ Scenario	1	2	3	4	5	6	7	8	9	Total
Random	✔	✔	✔	✔	✔	✔				6
Monkey	✔	✔					✔			3
QBEa	✔	✔			✔			✔		4
FARLEADa	✔	✔		✔	✔	✔	✔	✔		7
FARLEADb	✔	✔	✔	✔	✔	✔	✔	✔	✔	9
FARLEADc	✔	✔	✔	✔	✔	✔	✔	✔	✔	9

A Test Generation Engine is effective

Only if it generates

- A **witness** for the given scenario
- **At least once** in
  - 100 executions (50000 episodes max)

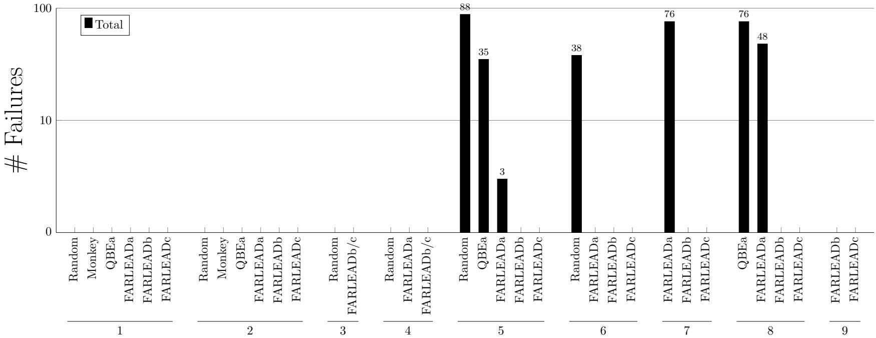
Effectiveness Results

Engine \ Scenario	1	2	3	4	5	6	7	8	9	Total
Random	✔	✔	✔	✔	✔	✔				6
Monkey	✔	✔					✔			3
QBEa	✔	✔			✔			✔		4
FARLEADa	✔	✔		✔	✔	✔	✔	✔		7
FARLEADb	✔	✔	✔	✔	✔	✔	✔	✔	✔	9
FARLEADc	✔	✔	✔	✔	✔	✔	✔	✔	✔	9

Overall, FARLEAD-Android is

- ✔ **More effective** than other engines.
  - **More effective** when **mixed** or **imperative** specs are used.

Number of Failures

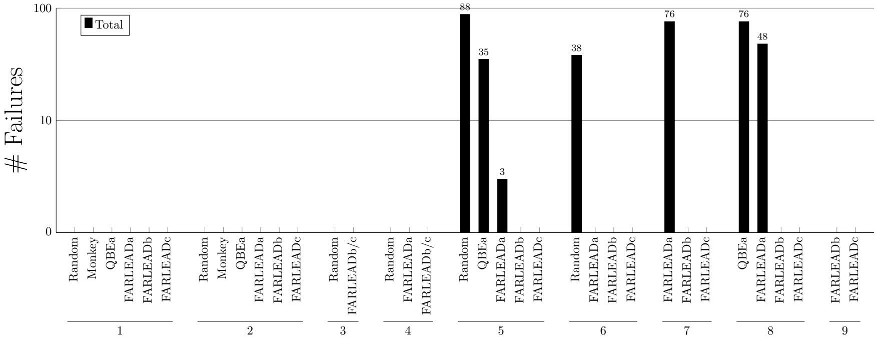


A Test Generation Engine fails

Only if it **cannot find**

- A **witness** for the given scenario
- **At least once** in an execution (500 episodes)

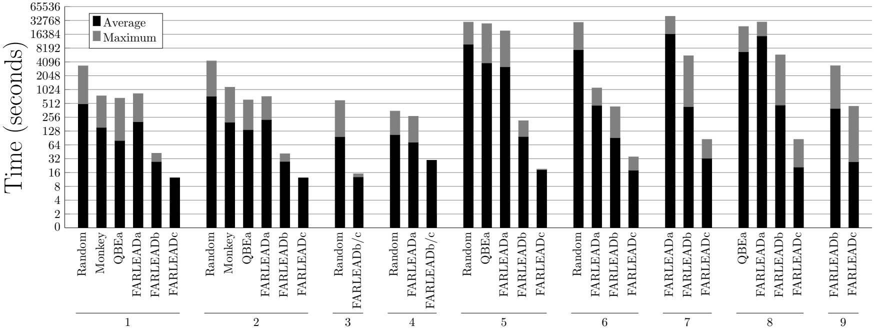
Number of Failures



Overall, FARLEAD-Android is

- ✔ **Fails fewer times** than other tools.
  - Does NOT fail when **mixed** or **imperative** specs are used.

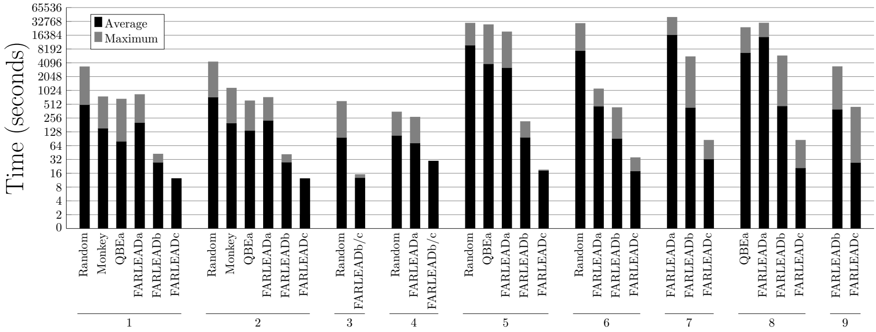
Test Generation Times



A Test Generation Engine Achieves Better Performance

Only if it **terminates faster** (generates a test).

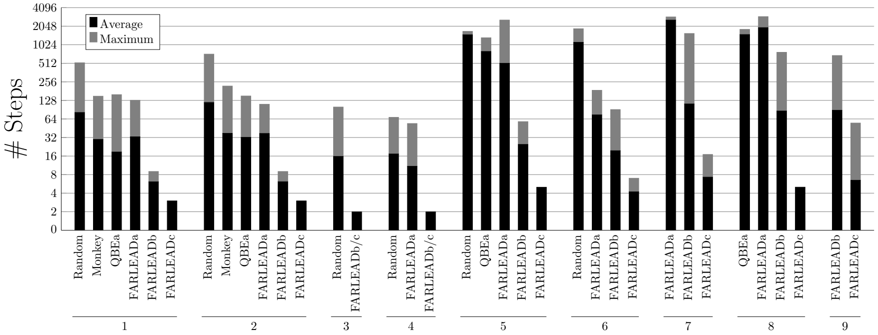
Test Generation Times



Overall, FARLEAD-Android is

- ✔ **Faster** than other tools.
  - Becomes faster from **declarative** to **imperative**.

Number of Steps



Typically, RL-LTL Engines

- ✖ Require **hundreds of thousands** steps (wait until convergence).
- ✔ FARLEAD-Android requires less than 4K steps.

Future Work #1: Gherkin Syntax

What is Gherkin Syntax?

Describes UI test scenarios.

- ✔ Customer **friendly** and **used in practice**.
- ✔ Easy to derive **from informal requirements**.

Syntax

Given p (precondition)
When q (antecedent)
Then r (consequent)

Example

Given The activity is TextNote
When The save button is clicked
Then A Note is created

Convert to LTL

- ✔ $\varphi = (\Diamond p) \wedge \bigcirc (p \, \mathcal{U} \, [q \wedge \bigcirc \Diamond r])$
- ? Natural language ⇒ atomic propositions? (**resolve ambiguity**)

Future Work #2: Bounded Metric Temporal Logic (BMTL)

Scenario #4: Non-functional Testing

- Developer **has to verify** that
  - The chess AI makes a move in less than 3 seconds.

✖ Impossible to describe in LTL.

Bounded Metric Temporal Logic

Describes

- **bounds** on the number of steps and
- **constraints** on the time required

Example

- $\varphi = \top \, \mathcal{U}_{[0,100]}^{[0,20]} \left(\text{userMoved} \wedge [\text{idle} \, \mathcal{U}_{[0,3]}^{\{1\}} \, \text{computerMoved}]\right)$

Miscellaneous Information

Related Paper

- Y. Koroglu and A. Sen, *Reinforcement Learning-Driven Test Generation for Android GUI Applications using Formal Specifications*, arXiv preprint, 2019.
- https://arxiv.org/abs/1911.05403
- Available in my webpage, see below.

Contact

- https://www.cmpe.boun.edu.tr/~yavuz.koroglu/
- yavuz.koroglu@boun.edu.tr
- DependLAB @ BM 21