Monkey See, Monkey Do: Effective Generation of GUI Tests with Inferred Macro Events

Markus Ermuth
Michael Pradel
TU Darmstadt
Motivation

How to test complex GUIs?

Manual  Analysis-based  Random
Motivation

How to test complex GUIs?

- Manual
- Analysis-based
- Random
- Realistic event sequences
- Huge effort
Motivation

How to test complex GUIs?

Manual  Analysis-based  Random

- Automatic
- Scalability issues
Motivation

How to test complex GUIs?

Manual | Analysis-based | Random

- Automatic and scalable
- Used in practice
Motivation

How to test complex GUIs?

Manual  Analysis-based  Random
- Automatic and scalable
- Used in practice
Problem

Header:
- Mouseover

Items:
- Mouseover
- Mouseout
- Click
Problem

- Effective testing requires complex, realistic sequences of events
- Probability to hit them by chance: Extremely small
Problem

- Effective testing requires complex, realistic sequences of events
- Probability to hit them by chance: Extremely small

Observation:
UI-level events ≠ Logical events
This Talk

Monkey see, monkey do

- **Learn** usage patterns from users
- **Imitate** them during test generation
Events vs. Macro Events

Event (implementation level)
- Type, target

Macro event (logical)
- Finite state machine
- Transitions = abstracted events
Overview

Record user actions

Infer macro events

Test generation
Overview

GUI application

Record user actions

Traces

Infer macro events

Macros

Test generation

Tests

Fully automatic
Recording User Actions

Trace: Sequence of events

... over, header over, item1 click, item1 out, item1 ...

... over, header over, item1 out, item1 over, item2 click, item2 out, item2 ...

etc.
Inference of Macro Events

Trace → Frequent subsequ. mining → Clustering → Finite automata inference → Macro events
Inference of Macro Events

Goal: Identify recurring patterns and remove noise

Adapted CloSpan algorithm [Yan et al., 2003]
- Bounded length of subsequences
- Structural relations between events
Inference of Macro Events

Frequent subsequ. mining

Clustering

Finite automata inference

Macro events

... over, header
... over, item1
... click, item1
... out, item1
... over, item2
... click, item2
... out, item2
Inference of Macro Events

Trace → Frequent subsequ. mining → Clustering → Finite automata inference → Macro events

... over, header over, item click, item out, item ...
... over, header over, item over, item click, item out, item ...
...
Inference of Macro Events

Trace → Frequent subsequ. mining → Clustering → Finite automata inference → Macro events

... over, header over, item click, item out, item ...

... over, header over, item out, item over, item out, item click, item out, item ...

...
Inference of Macro Events

Goal: Group related subsequences

Prefix clustering:
- Same initial event $\sim$ same cluster
Inference of Macro Events

Trace → Frequent subsequ. mining → Clustering → Finite automata inference → Macro events

over, header
over, item
click, item
out, item

over, header
over, item
out, item
click, item
out, item
Inference of Macro Events

Trace → Frequent subsequ. mining → Clustering → Finite automata inference → Macro events

over, header
over, item
click, item
out, item

over, header
over, item
out, item
over, item
click, item
out, item
Inference of Macro Events

Goal: **Summarize sequences into macros**

Adapted k-tails algorithm [Biermann, Feldman, 1972]
- Optimized state merging
- Structural relations between events
Inference of Macro Events

Trace → Frequent subsequ. mining → Clustering → Finite automata inference → Macro events

Finite automata inference
Test Generation

- Interleave random testing with macro replay
- Pick and replay macros based on available events
- Replay active macro until reaching a final state
More Details in the Paper

Monkey See, Monkey Do: Effective Generation of GUI Tests with Inferred Macro Events

Markus Ermuth
Department of Computer Science
TU Darmstadt, Germany
markus.ermuth@gmail.com

Michael Pradel
Department of Computer Science
TU Darmstadt, Germany
michael@binaervarianz.de

ABSTRACT

Automated testing is an important part of validating the behavior of software with complex graphical user interfaces, such as web, mobile, and desktop applications. Despite recent advances in UI-level test generation, existing approaches often fail to create complex sequences of events that represent realistic user interactions. As a result, these approaches cannot reach particular parts of the application under test, which then remain untested. This paper presents a UI-level test generation approach that exploits execution traces of human users to automatically create complex sequences of events that go beyond the recorded traces. The key idea is to infer so-called macro events, i.e., sequences of low-level UI events that correspond to a single logical step of interaction, such as choosing an item of a drop-down menu or filling and submitting a form. The approach builds upon and adapts well-known data mining techniques, in particular frequent subsequence mining and inference of finite state machines. We implement the approach for client-side web applications and apply it to four real-world applications. Our results show that our approach outperforms state-of-the-art approaches in terms of the quality of generated test cases. However, the complexity of many GUI applications makes manual UI-level testing difficult. For example, a complex client-side web application may consist of dozens of pages that each provide hundreds of events that a tester may trigger. Because exploring such programs manually is difficult, automated test generation approaches have been proposed [26, 24, 27, 11, 8, 42, 17, 35]. The basic idea is to generate sequences of UI events that achieve high coverage or that trigger a particular kind of problem. Existing approaches include black-box approaches, such as the popular Monkey runner for Android 1, which triggers random UI events, and white-box approaches, which, e.g., symbolically analyze the programs code to find events worth triggering.

Despite recent advances in UI-level test generation, two important challenges remain. First, deeply exploring a program often requires complex sequences of events. For example, consider a program that uses a drop-down menu to connect pages to each other. To reach another page, a test generator must move the mouse into the menu, wait until...
Implementation

Client-side web applications

Builds on WebAppWalker

- Framework for UI-level testing
- Firefox add-on
- Strategies for selecting events

https://github.com/michaelpradel/WebAppWalker/
Evaluation

Effectiveness and efficiency?

Setup:
- 4 real-world applications
- 16 usage traces
- Comparison with random testing
Visited Pages

How many pages do the generated tests reach?

Significant improvements for 3/4 applications
Branch Coverage

How many branches do the generated tests cover?

Significant improvements for 3/4 applications
 Covered Usage Scenarios

How many usage scenarios do the generated tests cover?

![Graph showing the number of tests covering usage scenarios]

- add content
- preview content
- comment article
- edit content
- delete content
- simple search
- advanced search
- user search

The graph above shows the number of tests covering each usage scenario. The scenarios are color-coded as follows:
- Green: macro
- Red: random

The y-axis represents the number of tests covering each usage scenario, while the x-axis lists the usage scenarios.
Performance

- Inferring macro events
  - 13 seconds – 85 minutes
  - One-time effort

- Test generation
  - 0.7 – 1.3 seconds per event
  - Only 8% slower than random testing
Future Work

- Cross-application macro learning
- Lightweight, in-production gathering of traces
- Scalability of inference algorithms
Conclusion

- **Macro events**: Abstract UI events into logical events
- **Infer and apply** macros: More effective GUI testing
- **Human knowledge improves automated testing**
Conclusion

- **Macro events:** Abstract UI events into logical events
- **Infer and apply** macros: More effective GUI testing
- **Human knowledge improves automated testing**

Thanks!