Automated Testing with Targeted Event Sequence Generation
Mobile Apps

● User-driven, event-based

● UI informs program state

● (Generally) low computational complexity/high # of event sequences
Previous test approaches

- Black box or model-based testing
- Symbolic execution testing from the start
- Backwards Breadth-First Search
Research Goal

Is there a reliable way to test event sequences and event handlers in an application such that any given target code can be reached?
The Two-Phase Algorithm

- Performs concolic execution to build summaries of event handlers
  - Symbolic Summarization
- Builds event sequences backwards from the target line to the start of the sequence using a UI-model (model-based testing)
  - Sequence Generation
TaxCollector Example

- TaxCollector Application
  - used to compute input tax for a given income amount

Figure 1: Two screens in *TaxCalculator*: (a) the income entry screen and (b) the result screen displaying the income, deductions, and resulting tax.
1 income = this.appState.enteredAmount;
2 deduction = 0;
3 if (Settings.getEnableTaxDeduction()) {
4     deduction = Settings.getTaxDeduction();
5 }
6 taxable = income - deduction;
7 if (taxable < 0) {
8     taxable = 0; // example target
9 }
10 tax = taxable * TAX_RATE;
11 result = income - tax;

Figure 2: UI model of a part of TaxCalculator.

Figure 3: Snippet from the onCreate method in the TaxResult activity in TaxCalculator.
Symbolic Summarization

● Preprocessing application code to produce event handler summaries for each handler
  ○ set of path summaries

● Computed using concolic execution

● May not cover all execution paths
  ○ mitigated by the small size of event handlers in mobile apps
Sequence Generation

● Generates test cases for target code snippets
  ○ Using the previously generated event handler summaries and the UI model
  ○ Loops through sequences to generate anchors and connectors
Constructing Anchors & Connectors

● Anchors
  ○ an execution path that the event handler sequence depends on
  ○ can be “pruned” away

● Connectors
  ○ generated from sequences between anchor and the event handler sequence
    ■ *must* correspond to acyclic path in UI model
Three Prioritization Heuristics

- Equivalent anchors
- Connector reprioritization
  - similar to equivalent anchors
- Increment-decrement reprioritization
  - ie: add/remove buttons
Research Questions

1. Is the algorithm able to generate test cases for challenging targets in real apps?

2. Does use of anchors and connectors affect ability to reach the targets?

3. Does prioritization heuristics affect ability to reach targets?
Implementation: Collider

- Collider implements this algorithm and uses it to analyze existing Android application codebases
  - TippyTipper, ConnectBot, Munchlife, OpenManager, DieDroid
## Collider Tests

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Targets depending on</th>
<th>Reached</th>
<th>Potential</th>
<th>Average size</th>
<th>Test case</th>
<th>Connectors</th>
<th>Pruning of anchors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequence</td>
<td>Parameters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TippyTipper</td>
<td>15</td>
<td>1</td>
<td>7</td>
<td>9</td>
<td>13</td>
<td>9</td>
<td>71%</td>
</tr>
<tr>
<td>ConnectBot</td>
<td>17</td>
<td>25</td>
<td>16</td>
<td>26</td>
<td>8</td>
<td>4</td>
<td>58%</td>
</tr>
<tr>
<td>Munchlife</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>29</td>
<td>7</td>
<td>66%</td>
</tr>
<tr>
<td>OpenManager</td>
<td>11</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td>39%</td>
</tr>
<tr>
<td>DieDroid</td>
<td>2</td>
<td>11</td>
<td>8</td>
<td>5</td>
<td>10</td>
<td>4</td>
<td>38%</td>
</tr>
</tbody>
</table>
Evaluation of Collider Results

● Symbolic Execution limitations
  ○ Strings and objects
  ○ High overhead for preprocessing (3-5 hrs)

● Sequence Generation benefits
  ○ Anchors, connectors reduce search space
  ○ Prioritization greatly increases efficiency + viability
    ■ 21 of 46 targets become unreachable w/out it
    ■ Remaining 25 increase runtime from 45 sec to 2.5 hrs
Questions
Questions

How can this be applied to other mediums such as web applications generally coded in scripting languages (ie: Javascript)?
Questions

Is testing the reachability of events in so few applications representative of the entire Android (or other application webstore) marketplace?
Questions

Are there other heuristics which could improve the runtime and efficiency of the sequence-building algorithm and the backwards path-finding algorithm?
Questions

Is the algorithm able to infer everything about the event handler summaries and the parameters necessary to reach the target?
Questions

How would the runtime increase as the UI model became more and more clutter
Questions

Could the event sequence generation algorithm be applied to game development in a meaningful way that would enable developers to create use-case diagrams?