Automatic Identification of Common and Special Object-Oriented Unit Tests

Tao Xie

Advisor: David Notkin
Dept. of Computer Science & Engineering
University of Washington, Seattle

Oct. 2004
Motivation

• Human loves writing unit tests!
  • Human capability is limited

• Machine comes to rescue!
  • Automated tools generate many test inputs
    Commercial: [Parasoft Jtest, Agitar Agitator, …]
    Academic:    [JCrasher@Gatech, Eclat@MIT,
                  Rostra@UW, Symstra@UW, …]
Problem

• Automated tools generate many test inputs
  • Infeasible to inspect all (6777 tests for LinkedList)
  • Select test inputs that throw exceptions or achieve new structural coverage
    [Parasoft Jtest, Agitar Agitator, JCrasher, …]

• Any “gold” left in the generated tests?
  • Need new gold mining devices
Sabicu: Automatic Identification of Common and Special Tests

- Intuition:
  common tests exercise common cases
  special tests exercise special cases

- Key:
  ways to characterize common and special cases

- Device:
  observe runtime behavior and infer
  universal properties: true all the time
  common properties: true most of the time

- Gold:
  - common tests: satisfy universal/common properties
  - special tests: violate common properties
Examples of Inferred Statistical Properties

- Syntactically identical to algebraic specs
- Universal property:
  \[
  \text{size(clear(S).state).retval} == 0
  \]
  Satisfying count: 121   Violating count: 0

- Common property:
  \[
  \text{remove(removeLast(S).state, m0_2).state} == \text{removeLast(remove(S, m0_2).state).state}
  \]
  Satisfying count: 318   Violating count: 42
Inferring Statistical Properties

During development of Sabicu

• Looked into human-written algebraic specs
• Predefined 25 abstraction templates [UW-CSE-04-08-03]

\[
\begin{align*}
\text{size(clear(S).state).retval} &= 0 \\
g(f(S, \text{args1}).state, \text{args2}).retval &= \text{const}
\end{align*}
\]

\[
\begin{align*}
\text{remove(removeLast(S).state, m0_2).state} \\
&= \text{removeLast(remove(S, m0_2).state).state} \\
g(f(S, \text{args1}).state, \text{args2}).state \\
&= f(g(S, \text{args1}).state, \text{args2}).state
\end{align*}
\]

At runtime

• Instantiate templates with instances (tests)
• Obtain statistical properties
Approach Overview

Rostra
[Xie et al. ASE 04]

Class bytecode

Test generation

Method-call composition

Statistical inference

Abstraction templates

Common or universal properties

Test identification

Special tests

Common tests
Subjects and Quantitative Results

http://www.cs.washington.edu/homes/taoxie/sobicu/

<table>
<thead>
<tr>
<th>subject</th>
<th>meth</th>
<th>axiom space</th>
<th>iter</th>
<th>axioms const</th>
<th>time (sec)</th>
<th>properties</th>
<th>tests</th>
<th>generated</th>
<th>special</th>
<th>common</th>
<th>both</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinSearchTree</td>
<td>4</td>
<td>240</td>
<td>3</td>
<td>75</td>
<td>0.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>75</td>
<td>1.32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>75</td>
<td>1.32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BinomialHeap</td>
<td>12</td>
<td>2364</td>
<td>3</td>
<td>501</td>
<td>44.36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>501</td>
<td>119.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>502</td>
<td>371.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FibonacciHeap</td>
<td>9</td>
<td>1242</td>
<td>3</td>
<td>287</td>
<td>1.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>287</td>
<td>3.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>287</td>
<td>7.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HashMap</td>
<td>13</td>
<td>2022</td>
<td>3</td>
<td>381</td>
<td>16.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>381</td>
<td>65.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>381</td>
<td>157.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HashSet</td>
<td>8</td>
<td>792</td>
<td>3</td>
<td>211</td>
<td>1.85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>211</td>
<td>2.65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>211</td>
<td>3.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LinkedList</td>
<td>21</td>
<td>6048</td>
<td>3</td>
<td>796</td>
<td>6.80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>797</td>
<td>21.88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>797</td>
<td>76.71</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SortedList</td>
<td>24</td>
<td>7827</td>
<td>3</td>
<td>877</td>
<td>10.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>878</td>
<td>33.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>878</td>
<td>110.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TreeMap</td>
<td>15</td>
<td>1968</td>
<td>3</td>
<td>409</td>
<td>20.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>409</td>
<td>79.94</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>409</td>
<td>314.46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntStack</td>
<td>4</td>
<td>252</td>
<td>3</td>
<td>33</td>
<td>0.57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>33</td>
<td>1.21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>33</td>
<td>2.84</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UBStack</td>
<td>10</td>
<td>1077</td>
<td>3</td>
<td>87</td>
<td>0.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>87</td>
<td>1.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>87</td>
<td>1.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More to be Done

More applications:
• Focused testing on universal properties
• Applied in software evolution
• Testing different implementations of the same interface

More evaluations
• Measure the fault detection capability and structural coverage of identified tests
• Case studies on programmers
Conclusion

• Statistical properties are useful too
  • Daikon: axiomatic spec inference [Ernst et al. TOSE 01]
  • Algebraic spec inference [Henkel&Diwan ECOOP 03]

• Gold mining is just starting:
  making the most out of generated tests!
  • Test selection based on operational violations [Xie&Notkin ASE 03]
  • Relating to industry: Agitar Agitator,
    Parasoft Jtest [Xie et al. ASE 04]
Examples of Common and Special Tests

removeLast(addFirst(S, m0_1).state).state
  == addFirst(removeLast(S).state, m0_1).state

Satisfying count: 117 (common test)
LinkedList m = new LinkedList();
m.add(0, new Integer(-1));
m.addFirst(new Integer(0));
m.removeLast();
Examples of Common and Special Tests

removeLast(addFirst(S, m0_1).state).state
    == addFirst(removeLast(S).state, m0_1).state

Satisfying count: 3 (special test)
LinkedList m = new LinkedList();
m.addFirst(new Integer(0));
m.removeLast();
Examples of Common and Special Tests

```java
remove(removeLast(S).state, m0_2).state
==removeLast(remove(S, m0_2).state).state
```

- Satisfying count: 318 (common test)
  ```java
  LinkedList m = new LinkedList();
  m.add(0, new Integer(-1));
  m.removeLast();
  m.remove(new Integer(-1));
  m.remove(new Integer(-1));
  ```
Examples of Common and Special Tests

\[
\text{remove}(\text{removeLast}(S).\text{state}, \ m0\_2).\text{state} \\
\text{==removeLast}(\text{remove}(S, \ m0\_2).\text{state}).\text{state}
\]

• Violating count: 42 (special test)
  LinkedList m = new LinkedList();
  m.add (0, new Integer(-1));
  m.add (0, new Integer(0));
  m.removeLast();
  m.remove(new Integer(-1));
Examples of Common and Special Tests

```
lastIndexOf(addFirst(S, m0_1).state, m0_2).retval
== (lastIndexOf(S, m0_2).retval + 1)
[where (m0_1==m0_2)]
```

- Satisfying count: 120 (special test)
  ```
  LinkedList m = new LinkedList();
  m.addFirst(new Integer(-1));
  m.lastIndexOf(new Integer(-1));
  m.lastIndexOf(new Integer(-1));
  ```