LCT: An Open Source Concolic Testing Tool for Java Programs

Kari Kähkönen, Tuomas Launiainen, Olli Saarikivi, Janne Kauttio, Keijo Heljanko and Ilkka Niemelä

BYTECODE 2011
Overview

• Concolic Testing
• Tool Overview
• Experiments
• Tool Demonstration
Concolic Testing

• Concolic testing (dynamic symbolic execution) is an automated testing method
  – Generate test inputs
  – Execute program with these inputs
  – Catch runtime errors (uncaught exceptions, assertion violations)

• Can we cover all the reachable statements with the tests?
  – E.g., random testing can have a very low probability on reaching certain statements
  – Concolic testing: Attempt to explore all feasible execution paths
Concolic Testing

- Concolic testing combines concrete and symbolic execution
  - Program is instrumented with additional statements to enable symbolic execution
  - Concrete execution guarantees that all the found bugs are real
- Symbolic execution collects path constraints that can be used to compute new test inputs that explore previously unexplored execution paths
- Path constraint are typically solved using SMT-solvers
Example

```c
int x = input();
if (x > 10) {
    x = x + 5;
    if (x == 50)
        error;
}
```

Path constraint is a conjunction of constraints along a path from root of the tree to a leaf node.
LCT – LIME Concolic Tester

- An open source concolic testing tool for sequential Java programs
- Instruments the program under test using Soot
- Uses Boolector for bit-precise constraint solving
  - For example, overflows and modulo-operator are handled precisely
- Supports distributed testing by allowing several tests to be executed in parallel
- Reports uncaught exceptions as errors
- Several related tools exist: CUTE/jCUTE, Pex, Klee,…
Tool Architecture

Java

Class Foo {
    int x;
    int y;
    void main (Arg

List of generated test inputs

Instrumenter

Byte code – Jimple – Byte code

Byte Code

Test Executor

Constraint solver

Test Selector

Test Executor

Constraint solver

Test Executor

Constraint solver

Test Executor

Constraint solver
Distributed Testing

- Concolic testing suffers from path explosion problem
- Testing separate execution paths can be done independently
  - Keep track of all the unexplored branched in the execution tree
  - Distribute the path constraints related to these branches to test executors
  - Solving path constraints centrally could cause a performance bottleneck
- Distributed testing allows taking advantage of multicore architectures and networks of computers
Limitations

- Java core classes can be problematic to instrument directly
  - LCT replaces some of the core classes with custom implemented counterparts that can be instrumented
- If the program under test contains un-instrumented classes, full path coverage cannot be guaranteed
- Floating point input values are not supported as the constraint solver does not support floating points
- LCT makes a non-alising assumption
  - \( A[i] = 0; A[j] = 1; \text{if} \quad (A[i] \neq 0) \text{ERROR}; \)
Experiments

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Paths</th>
<th>1 executor</th>
<th>10 executors</th>
<th>20 executors</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVL tree</td>
<td>3840</td>
<td>16m 57s</td>
<td>2m 6s / 8.1</td>
<td>1m 8s / 15.0</td>
</tr>
<tr>
<td>Quicksort (5)</td>
<td>514</td>
<td>3m 11s</td>
<td>21s / 5.2</td>
<td>13s / 8.4</td>
</tr>
<tr>
<td>Quicksort (6)</td>
<td>4683</td>
<td>28m 22s</td>
<td>3m 29s / 8.1</td>
<td>1m 39s / 17.2</td>
</tr>
<tr>
<td>GCD</td>
<td>2070</td>
<td>11m 12s</td>
<td>1m 13s / 9.2</td>
<td>38s / 17.7</td>
</tr>
</tbody>
</table>

• The distributed nature of LCT has been evaluated by testing Java programs with varying number of test executors running concurrently
Experiments

<table>
<thead>
<tr>
<th>Approach</th>
<th>1-bounded</th>
<th>2-bounded</th>
<th>3-bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoupled</td>
<td>121 (54.50%)</td>
<td>185 (83.33%)</td>
<td>221 (99.95%)</td>
</tr>
<tr>
<td>Coupled</td>
<td>123 (55.41%)</td>
<td>187 (84.23%)</td>
<td>221 (99.95%)</td>
</tr>
<tr>
<td>Random</td>
<td>95 (42.79%)</td>
<td>151 (68.02%)</td>
<td>184 (82.88%)</td>
</tr>
</tbody>
</table>

• LCT has been used in a case study to compare random testing and concolic testing (SPIN 2010)
• Here LCT was used on a large number of mutants of a Java Card application to discover if the mutations changed the behavior of the program
Future Work

• We are currently extending LCT to support testing of multi-threaded Java programs
  – Support for multi-threaded programs will be released soon in LCT 2.0

• Support for C language based on the LLVM compiler infrastructure is also in development

• We are investigating how to support incremental testing by exploring only execution paths affected by recent changes
Availability

• LCT is open source and available from http://www.tcs.hut.fi/Software/lime