Generating Asserts for Test Cases Efficiently

Student: Luís Pinho

Supervisor: Rui Maranhão (PhD)
Co-Supervisor: José Campos (MSc)

10th July, 2013
Agenda

- Context
- Motivation
- State of the art
- LEOPArD
- Evaluation
- Conclusions
- Future Work
- Contributions
Context
Motivation
Motivation

Therac-25

- At least 5 patients died, while several others became seriously injured.

- A race-condition caused the machine to give approximately 100 times the intended dose, causing radiation overdose.
The explosion of Ariane 5:

- The explosion occurred after 37 seconds of start (30 seconds after lift-off).
- The exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed Integer value.
Motivation

A problem has been detected and Windows has been shut down to prevent damage to your computer.

DRIVER_IRQL_NOT_LESS_OR_EQUAL

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x000000D1 (0x0000000C, 0x00000002, 0x00000000, 0xF86B5A89)

*** gv3.sys - Address F86B5A89 base at F86B5000, DateStamp 3dd991eb

Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further assistance.
Motivation

Fully Automated Test Case Generation
Software Development Cycle
The Debug Phase
Automatic Debugging Tools

GZoltar

Tarantula
Spectrum-based fault localization (SFL)
An Invariant is a data structure that is valid during the execution of a program.

**Range Invariant**
Stores the values observed in a range structure.

\[ 10 < x < 20 \]
("x" is valid between 10 and 20)

**Bloom Filters**
Stores the values observed in a set structure.

\[ x = \{10, 20, 30\} \]
("x" is valid for the values 10, 20 and 30)
GZoltar

- Detect all Open Projects
- Detect Test Classes
- Execute Test and obtain its Result
- Instrument all Classes
- Obtain Coverage Result
- Create Coverage Matrix Line
- Create Constraints
- Execute the MINION Constraint Solver
- Obtain all solutions
- Filter solutions
- Show solutions and Code Editor Integration

for each open project
for each test class
for each line
Goal
public class Add extends TestCase{

        @Test
        public void testAdd() {
            int result = Calculator.add(1, 2);
            assertTrue(result == 3);
        }
    }
How?
Dynamically Inferring Behavioural Models with Fault Screeners

LEOPArD

Wait For Start

Play

Stuck

Game Over
Training Phase

- LEOPArD learns the expected behaviour of a software program using correct executions in a so-called training phase.

- The Finite State Machine is created.

- The Invariants are initialized with primary types.
Operational Phase

- After the model is learnt and all invariants are trained, each execution during the operational phase is checked against it for consistency.

- LEOPArD uses the behavioural model learnt to act as an oracle to decide the pass/fail of an execution.
First step - LEOPARD

<table>
<thead>
<tr>
<th>TEST 1</th>
<th>add(1, 2) -&gt; sub(2, 1) -&gt; add(2, 3) -&gt; mult(4, 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>=3 -&gt; =2 -&gt; =5 -&gt; =8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEST 2</th>
<th>mult(3, 2) -&gt; div(2, 1) -&gt; add(8, 2) -&gt; sub(2, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>=6 -&gt; =2 -&gt; =10 -&gt; =2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEST 3</th>
<th>mult(3, 2) -&gt; div(2, 1) -&gt; sub(2, 1) -&gt; mult(2, 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>=6 -&gt; =2 -&gt; =1 -&gt; =6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEST 4</th>
<th>add(2, 3) -&gt; sub(3, 2) -&gt; add(3, 4) -&gt; mult(5, 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>=5 -&gt; =1 -&gt; =7 -&gt; =15</td>
</tr>
</tbody>
</table>
Second step - LEOPARDo

TEST 1
add(1,2) -> sub(2,1) -> add(2,3) -> mult(4,2)
  =3    =2    =5    =8

TEST 2
mult(3,2) -> div(2,1) -> add(8,2) -> sub(2,0)
  =6    =2    =10   =2

TEST 3
mult(3,2) -> div(2,1) -> sub(2,1) -> mult(2,3)
  =6    =2    =1    =6

TEST 4
add(2,3) -> sub(3,2) -> add(3,4) -> mult(5,3)
  =5    =1    =7    =15
Third step - LEOPArD

State Machine

Diagram showing a state machine with states and transitions labeled with function calls such as `functions.Calculadora add [1, 2]` and `functions.Calculadora sub [2, 1]`.
Third step (+1) - LEOPArD

State Machine + Equivalence (k=2)
Evaluation
Evaluation (Last Step) - LEOPArD

P <- INJECT FAULT IN P

for x in \{10,30,50,70,90,100\} do
    for i in [1..20] do
        T' <- SELECT x passing test cases from T
        FSM_B <- LEARN (P,T')
        if OperationalPhase then
            ResultsTEMP += CollectResultsFromP (Fp, Fn, SFL)
        end if
    end for
end for

Results += AVERAGE (ResultsTEMP)
## Experimental Setup

<table>
<thead>
<tr>
<th>Subject</th>
<th>Version</th>
<th>LOCs</th>
<th>Test Cases</th>
<th>Coverage</th>
<th>Number of States</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>0.1</td>
<td>2418</td>
<td>20</td>
<td>14.8%</td>
<td>61</td>
</tr>
<tr>
<td>NanoXML</td>
<td>2.2.3</td>
<td>5393</td>
<td>9</td>
<td>45.7%</td>
<td>6050</td>
</tr>
<tr>
<td>org.jacoco.report</td>
<td>0.5.7</td>
<td>5979</td>
<td>99</td>
<td>73.1%</td>
<td>763</td>
</tr>
</tbody>
</table>

- **Single Bug** (3)
- **Double Bug** (3)
- **Triple Bug** (1)
Experimental Setup

- Check the following cases:
  - The sequence doesn't exist -> FAIL
  - The sequence exists but the ranges were violated -> FAIL
  - The sequence exists and the ranges were accepted -> PASS
Evaluation Metrics

- False Positive Rate
  \[ f_p := \frac{N_{fp}}{N_t} \]

- False Negative Rate
  \[ f_n := \frac{N_{fn}}{N_t} \]

- Cost of diagnosis
  \[ C_d = \frac{|\{i|s_o(i) > s_o(d^*)\}| + \frac{|\{i|s_o(i) \geq s_o(d^*)\}|}{2} - 1}{|\{i|s_o(i) \geq s_o(d^*)\}|}, \quad 1 \leq i \leq M \]
Results - Single Bug

(a) ATM single fault.

(b) NanoXML single fault.

(c) org.jacoco.report single fault.
Results - Double Bug

(a) ATM double fault.
(b) NanoXML double fault.
(c) org.jacoco.report double fault.
Results - Triple Bug

(a) ATM triple fault.

(b) NanoXML triple fault.

(c) org.jacoco.report triple fault.
Conclusions
Conclusions

● LEOPArD yields similar diagnostic quality, while entailing acceptable runtime overhead.

● With only 30% of the total passing test cases, we can obtain similar diagnostic accuracy (only 2 more statements that need inspection), despite entailing an average of 8 false positives and 1 false negative.
Future Work

- Monitoring of non-primitive variables. (String, char...)

- Tune the behavioural model in order to decrease the slowdown

- Algorithms to reduce the state machine size
Contributions

- IJUP'13 <- Accepted
- ICTSS'13 <- Submitted and under review
Discussion