Incremental Learning-Based Testing for Reactive Systems

Karl Meinke, Muddassar Sindhu
Royal Institute of Technology (KTH)
Stockholm
0. Overview of Talk

1. Specification Based Black-box Testing

2. Learning Based Testing paradigm (LBT)
   - connections between learning and testing
   - testing as a search problem
   - testing as an identification problem
   - testing as a parameter inference problem

3. Chosen Framework: reactive systems

4. Results

5. Conclusions
1. Specification Based Black-Box Testing

1. System requirement \((\text{Sys-Req})\)
2. System under Test \((\text{SUT})\)
3. Test verdict \text{pass/fail} \(\text{(Oracle step)}\)

Diagram:

- **TCG**
  - Test case
  - Constraint solver

- **SUT**
  - Language runtime

- **Oracle**
  - Output
  - Constraint checker

Sys-Req and pass/fail arrows connect the components.
1.1. Procedural System Example: *Newton’s Square Root Algorithm*

**precondition** $x \geq 0.0$

TCG
Constraint solver

SUT
Newton Code

Oracle
Constraint checker

$x=4.0$ satisfies $x \geq 0.0$

Postcondition
$\mid y^2 - x \mid \leq \varepsilon$

Verdict
$x=4.0$, $y=2.0$ satisfies
$\mid y^2 - x \mid \leq \varepsilon$
1.4. Reactive System Example: Coffee Machine

Sys-Req: always( in=\$1 \text{ implies after}(10, \text{ out}=\text{coffee}) ) \quad \text{pass/fail}

\begin{align*}
in_0 &= \$1 \\
out_{11} &= \text{coffee}
\end{align*}

\text{Satisfies}
\[\text{always( } \text{in}=1\$ \text{ implies after}(10, \text{ out}=\text{coffee})\text{)}\]
1.2. Key Problem: Feedback

**Problem:** How to modify this architecture to..

1. Improve next test case using previous test outcomes
2. Execute a large number of good quality tests?
3. Obtain good coverage?
4. Find bugs quickly?
2. Learning-Based Testing

“Model based testing without a model”
2.1. Basic Idea ...

LBT is a search heuristic that:

1. Incrementally learns an SUT model
2. Uses generalisation to predict bugs
3. Uses best prediction as next test case
4. Refines model according to test outcome
2.2. Abstract LBT Algorithm

1. Use \((i_1, o_1), \ldots, (i_k, o_k)\) to learn model \(M_k\)
2. Model check \(M_k\) against \(Sys-Req\)
3. Choose “best counterexample” \(i_{k+1}\) from step 2
4. Execute \(i_{k+1}\) on SUT to produce \(o_{k+1}\)
5. Check if \((i_{k+1}, o_{k+1})\) satisfies \(Sys-Req\)
   a) Yes: terminate with \(i_{k+1}\) as a bug
   b) No: goto step 1

Difficulties lie in the technical details ...
2.3. General Problems

Difficulty is to find combinations of models, requirements languages and Sat algorithms \((M, L, A)\)

so that ...

1. models \(M\) are:
   - expressive,
   - compact,
   - partial and/or local (an abstraction method)
   - easy to manipulate and learn

2. \(M\) and \(L\) are feasible to model check with \(A\)
3. Chosen Framework for Study:

1. SUT = reactive system
2. Model = deterministic Kripke structure
3. Sys-Req Lang = linear temporal logic (LTL)
4. Learning = IKL incremental learning algorithm
5. Model Checker = NuSMV
LBT Architecture

Flowchart:
- Random Input Generator
- NuSMV Model Checker
  - NuSMV Model Checker
  - Hypothesis Automata $M_n$
  - Hypothesis Automata $M_n$
- Hypothesis Automata $M_n$
- Oracle $\bar{p} = \bar{o}$
- Oracle $\bar{p} = \bar{o}$
- IKL Algorithm
- IKL Algorithm
- SUT
  - input $\bar{i}$
  - observed output $\bar{o}$
- LTL requirement $\phi$
- Equivalence Checker $SUT = M_n$
  - Equivalence Checker $SUT = M_n$
- true/stop
- No $\bar{i} \in MCQ$
- Yes $\bar{i} \in MCQ$
- fail, warning, stop
- pass/$\bar{i}, \bar{o}$
A Case Study: Elevator Model
### Elevator Results

<table>
<thead>
<tr>
<th>Req</th>
<th>$t_{first}$ (sec)</th>
<th>$t_{total}$ (sec)</th>
<th>MCQ first</th>
<th>MCQ tot</th>
<th>PQ first</th>
<th>PQ tot</th>
<th>RQ first</th>
<th>RQ tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req 1</td>
<td>0.34</td>
<td>1301.3</td>
<td>1.9</td>
<td>81.7</td>
<td>1574</td>
<td>729570</td>
<td>1.9</td>
<td>89.5</td>
</tr>
<tr>
<td>Req 2</td>
<td>0.49</td>
<td>1146</td>
<td>3.9</td>
<td>99.6</td>
<td>2350</td>
<td>238311</td>
<td>2.9</td>
<td>98.6</td>
</tr>
<tr>
<td>Req 3</td>
<td>0.94</td>
<td>525</td>
<td>1.6</td>
<td>21.7</td>
<td>6475</td>
<td>172861</td>
<td>5.7</td>
<td>70.4</td>
</tr>
<tr>
<td>Req 4</td>
<td>0.052</td>
<td>1458</td>
<td>1.0</td>
<td>90.3</td>
<td>15</td>
<td>450233</td>
<td>0.0</td>
<td>91</td>
</tr>
<tr>
<td>Req 5</td>
<td>77.48</td>
<td>2275</td>
<td>1.2</td>
<td>78.3</td>
<td>79769</td>
<td>368721</td>
<td>20.5</td>
<td>100.3</td>
</tr>
<tr>
<td>Req 6</td>
<td>90.6</td>
<td>1301</td>
<td>2.0</td>
<td>60.9</td>
<td>129384</td>
<td>422462</td>
<td>26.1</td>
<td>85.4</td>
</tr>
</tbody>
</table>
5. Conclusions

• A promising approach ...
• Flexible general heuristic,
  • many models and requirement languages seem possible
• Many SUT types might be testable
  • procedural, reactive, real-time etc.

Open Questions

• Benchmarking?
• Scalability? (abstraction, infinite state?)
• Efficiency? (model checking and learning?)