HSST 2015
Learning-Based Testing for Procedural and Reactive Systems

Karl Meinke, karlm@kth.se

School of Computer Science and Communication
KTH Royal Institute of Technology Stockholm
0. Overview of the Course

Part 1: Introduction to Learning-based Testing
1. Requirements Based Black-box Testing
2. Learning Based Testing Paradigm (LBT)
3. Two Frameworks for Study

Overview

Part 2: LBT for reactive systems: theory

Part 3: LBT for reactive systems: praxis

Part 4: LBT for procedural systems
1. Requirements Based Black-Box Testing

1. User requirement  \( SUT-\text{Req} \)
2. System under Test  \( SUT \)
3. Test verdict pass/fail  \( Oracle \)
1.1. Procedural Code Example: *Newton’s Square Root Algorithm*

**Precondition** $x \geq 0.0$

**Postcondition** $|y^2 - x| \leq \varepsilon$

---

**TCG**

- Constraint solver

**SUT**

- Newton Code

**Oracle**

- Constraint checker

---

Input: $x=4.0$

Output: $y=2.0$

Verdict:

$x=4.0, y=2.0$ satisfies $|y^2 - x| \leq \varepsilon$

---

$x=4.0$ satisfies $x \geq 0.0$
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 (LBT)
Meinke 2004, Proc. ISSTA-04

“aka. Model based testing without a model”
LBT is a search heuristic that:

1. Partially and incrementally learns an SUT model
2. Uses generalisation (*inductive inference*) to predict unseen bugs!
3. Uses best prediction as next test case
4. Iteratively refines model according to each test outcome
2.2. Abstract LBT Algorithm

1. Start from *null hypothesis* $M_0$
2. For each $k \geq 0$ do
   1. Model check $M_k$ against *SUT-Req*
   2. Choose “best counterexample” $i_{k+1}$ from step 2.1
   3. Execute $i_{k+1}$ on SUT to produce $o_{k+1}$
   4. If $(i_{k+1}, o_{k+1})$ satisfies $\neg SUT-Req$ label $i_{k+1}$ as a bug
   5. Use $(i_{k+1}, o_{k+1})$ to refine $M_k$ to $M_{k+1}$
   6. If *finished* break.

When Step 2.2 fails we fall back on:
   – Active learning queries
   – Equivalence checking queries
2.3. Technical Difficulties

General problem is to find combinations of models, requirements languages and solvers \((M, L, S)\) so that ...

1. models \(M\) are:
   - expressive,
   - compact,
   - partial and/or local (an abstraction method)
   - easy to construct and learn
   - behaviour is captured by \(L\)

2. \(M\) and \(L\) are feasible to model check with \(S\)

3. Supervised learning of \(M\) admits a notion of convergence
2.4. Convergence and Test Case Choice

- How reliable are counterexamples $c_1, ..., c_n$?
- Question of false negatives
- Some (parts of) SUTs more easily learned than others
- Measure local convergence around model points
- Convergence is a proxy for model reliability ...

“Counterexamples from locally well-converged regions are more reliable”
2.5. Convergence and Coverage

• Convergence is also proxy for coverage
• If no counterexamples ( \( n = 0 \) )
  – choose point from least converged region
    (breadth first search)

• **Question**: Do formal models of approximation and convergence always exist?
• **Answer**: sometimes, but important exceptions also exist.

Generally **data-oriented testing**

1. **Requirements Language** – pre and postconditions
   – first-order logic of real-closed fields

2. **Models**
   – non-gridded $n$-dimensional piecewise polynomials

3. **Model checker**
   – Hoon-Collins CAD algorithm, (Mathematica)

4. **Learning algorithm**
   – $n$-dimensional polynomial interpolation
Framework 2: Reactive Systems

Generally control-oriented testing

1. Requirements language = propositional linear temporal logic (PLTL)
2. Model = FSM, Moore machine
3. Model checker = BDD/SAT-based checkers
4. Learning = regular inference algorithms
Why not Neural Networks?

Neural and deep neural networks have notable recent success ... but several problems here ...

1. NN are implicit continuous models unsuited to symbolic model checking
2. NN learning paradigm based on iterative training (weight optimisation) on big data
   Testing does not fit this paradigm
   Single test case can take 1-10 minutes!
3. NN models are statistical in character
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, hybrid etc.

Open Questions

• Benchmarking?
• Scalability? abstraction, dimension reduction?
• Bottlenecks? model checking, learning, SUT?