LEARNING-BASED TESTING FOR REACTIVE SYSTEMS

Karl Meinke
CSC School
KTH Stockholm
Overview of Talk

1. What is learning-based testing?
2. The LBTest tool
3. Automotive Case Studies
4. Convergence Metrics
5. Recent Research
6. Conclusions

Unifying theme of 3, 4 and 5 is measuring model convergence since we rarely learn to completion.
1. What is Learning-based testing?

**Example:** black-box TCG for Newton’s algorithm

- **Precondition** \( x \geq 0.0 \)
- **Postcondition** \( |y^2 - x| \leq \varepsilon \)

Constraint solver → TCG → SUT → Oracle

- **Input** \( x=4.0 \)
- **Output** \( y=2.0 \)

**Verdict**

- \( x=4.0, y=2.0 \) satisfies \( |y^2 - x| \leq \varepsilon \)
1.2. Key Problem: **No 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”
Abstract LBT Algorithm

1. \( M_0 := \text{getInitialHypothesis}(); \)

2. For each \( k \geq 0 \) do
   1. Model check \( M_k \) against \( \text{Req} \)
   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 \( \neg \text{Req} \) label \( i_{k+1} \) as an error
   5. If equivalent \( (SUT, M_{k+1}, \text{Bound}) \) break.
   6. \( M_{k+1} := \text{getNextHypothesis}(i_{k+1}, o_{k+1}) \)
2. LBTest Tool

- LBTest implements *black-box requirements testing* for *embedded systems* with *off-the-shelf* and *customised components*.

- LBTest automates 3 processes:
  - **Test Case Generation** (ATCG), 3 Sources:
    - Active learning queries
    - Model checker generated counterexamples
    - Stochastic equivalence checker queries
  - **Test execution** (online testing)
  - **Verdict construction** *(pass/fail/warning/exception)*

- Some configurations quickly achieve *high model convergence*. 
LBTest Architecture

Communication wrapper

System under Test
- e.g. jar file

Automaton Learning Algorithm

NuSMV Model Checker
- TCG and Oracle

LTL Requirement Formula $Req$

Verdict $v_n$

observed output $o_n$

test case $i_n$

$n = 1, 2, \ldots$

Stochastic equivalence checker

counterexample $i_n$
Technical & Process Advantages

• Well suited to agile development
• Model is always synchronised to actual code
• No false positives or false negatives due to wrong/outdated models (C.f. model-based testing)
• Avoid manual model construction and maintenance
Modular Structure

- **Learners**
  - L*Mealy
  - Kearn’s algorithm
  - CGE, ICGE (term rewriting system representation)
  - MinSplit (NDFA representation)
  - Hybrid automaton learner HyCGE (infinite state systems)

- **Model checkers**
  - NuSMV 2.5
    - BDD checker
    - BMC/SAT solver
  - nuXmv 1.0

- **Stochastic equivalence checker**
  - First / longest / shortest difference (strategies)
Requirements Modeling

- Modeling reactive systems needs a **time concept**
- LBTest uses *propositional linear temporal logic* (PLTL)
- PLTL = “Boolean logic + time”
- Conventional **model-based testing (conformance testing)** is the *next-only part* of PLTL.

- Could interface LTL to *visual requirements modeling languages* and *pattern languages*. 
Approximate Models

- Real-world SUTs are \textit{infinite state systems}.
- LBTest constructs finite state approximations through \textit{finite partition sets}.

- Input partitioning is implemented in LBTest (test selection).
- Output partitioning is implemented in SUT wrapper (equivalence class).
- Gives a limited \textit{first-order extension} to PLTL.
Verdict Construction (Oracle step)

- On-the-fly verdict construction filters false negatives

- Compares two behaviours:
  (1) a predicted behaviour from model (bad)
  (2) an observed behaviour in SUT

- Prediction == Observation  =>  Fail/Warning
- Prediction != Observation  =>  Pass
- No Observation  =>  Exception/Timeout error
3. Automotive Case Studies (Volvo, Scania)

- **Engine Start**: 31 states, 220 transitions, 5 min.

- **Dual-circuit Steering**: 60 states, 800 transitions, 7.5 hours

- **Fuel level display**: 26 states, 104 transitions, 2.5 hours

- **Brake-by-wire**: 85,000 states, 1.7 million transitions, 10 hours
Case Study: Brake-by-Wire ECU
Fourteen Black-box Requirements

REQ-4 If the brake pedal is pressed and the actual speed of the vehicle is larger than 10 km/h and the slippage sensor shows that the (front right) wheel is slipping, this implies that the corresponding brake torque at the (front right) wheel should very quickly be 0.

\[ G( \text{BrakePedal} = b \land \text{Motion} = \text{moving} \land \text{SlipRR} = \text{slipping} \rightarrow X( \text{ABSBrakeTorqueRR} = \text{zero} ) ) \]
Model #3 after 400 msec
4. Convergence metrics

• LBT needs a stopping criterion

• We can try to use model convergence values

• Based on discrepancy
  • Difference between two successive models $M_i, M_{i+1}$
  • Difference between a model $M_i$ and the SUT

• Different metrics are possible:
  • Percentage of divergent sequences
  • Mean time to divergence

• Can estimate these by Monte Carlo methods
  • stochastic equivalence checking
Divergent Path Metric

- A fixed (user-defined) number $K$ of random input sequences is executed both on the learned model and on the SUT. ($K = \text{sample size}$)

- Input length is twice active learning length.

- The number $\delta$ of such random input sequences that show any divergent behavior between the model and the SUT is counted, where $0 \leq \delta \leq K$.

- The final divergent path metric $\text{DPM}$ is then the normalised percentage $\text{DPM} = 100 \times (K - \delta)/K$ (%).
Fig. 3. BBW: model convergence over time.

Fig. 4. BBW: model size over time.
5. Regular Inference by Over-Approximation

an equivalence relation \(\equiv \subseteq Q \times Q\) on the state set of a Moore automaton

\[
A = \langle Q, \Sigma, \Omega, \delta : Q \times \Sigma \rightarrow Q, \lambda : Q \rightarrow \Omega, q_0 \rangle
\]

is a congruence if, and only if, \(\equiv\) satisfies the substitutivity conditions

\[
q \equiv q' \rightarrow \delta( q, \sigma ) \equiv \delta( q', \sigma ), \tag{1}
\]

\[
q \equiv q' \rightarrow \lambda( q ) = \lambda( q' ), \tag{2}
\]

To achieve determinism, non-equivalences between states are propagated through the set of all observed input strings, by inverting rule (1) to its contrapositive form

\[
\delta( \overline{\sigma_1}, \sigma ) \neq \delta( \overline{\sigma_2}, \sigma ) \rightarrow \overline{\sigma_1} \neq \overline{\sigma_2} \tag{3}
\]

for input strings \(\overline{\sigma_1}, \overline{\sigma_2} \in \Sigma^*\).
Quantifying the Complexity of Testing and Learning
6. Conclusions

- Advantages of LBT
  - Flexible (black-box)
  - High-volume, high coverage (active learning)
  - More effective than random testing
  - Objective measurable coverage (stochastic equivalence checking)
  - Connects to PAC learning
  - Accurate test verdicts (model checking)
  - Supports an ALARP principle for safety standards?
Future Research

• Latency problems – concurrent testing and learning.
• Fault injection on virtualised hardware models.
• Hardware-in-the-loop (HIL) testing.
• Hybrid and infinite state systems.
• Testing systems-of-systems (SafeCOP)
Literature
