Online Testing of Real-time Systems Using UPPAAL

Kim G. Larsen, Marius Mikučonis, Brian Nielsen
{kgl, marius, bnielsen}@cs.aau.dk

Outline
Online testing using UPPAAL: NWPT03, FATES'04, EMSOFT'05.
- Formal framework of timed conformance testing of black-box.
  - Classical model-based black-box testing.
  - Test setup: from system and specification to testing.
  - Relativized timed input/output conformance relation.
  - Ordering of environments by discriminating power.
- Online Real-time Test Generation
  - Symbolic techniques from UPPAAL.
  - Online testing algorithm animated.
  - Real-time mapping to model and back.
- Evaluation: performance, industrial study, light controller demo.
- Conclusions and future work.

Motivation for Automated Testing

- What is testing?
  - checking the quality (functionality, reliability, ...) of an object
  - by performing experiments
  - in a controlled (and systematic) way.
- Testing is the main validation technique used by industry:
  - 10-20 errors per 1000 lines of code.
  - 30-50% of development time and cost in embedded software.
- Testing is still ad-hoc, based on heuristics, and error prone.
- Testing is routine, tedious and boring work – let machines do it.
- Not! Testing requires most of development skills.
- Verification vs. testing: abstract models vs. real world.
- Conformance testing is undecidable.

Related Work
This work is based on the following ideas:
- UPPAAL model checking algorithms for timed systems (1994).
- Jan Tretmans' testing theory (un-timed, quiescent) (1999).
- TorK testing tool framework (un-timed, w/o environment) (2000).

Test Setup: System ⇒ Model ⇒ Online Testing
Imp is (weakly) input enabled.
Clear and explicit Env assumptions.
Imp|Env forms a closed system.
Observable input/output actions.

Testing with general Env is expensive and often unnecessary.
Flexible: only relevant behavior (Env change, guiding, debug).

Environment ordering: f is more discriminating than e:

Online generation allows long and otherwise exhaustive tests.
Test Specification: Timed Automata Networks

Timed Automaton over $A$ is $\langle L, l_0, X, D, E, I \rangle$:
- $L$: set of locations,
- $l_0 \in L$: the initial location,
- $X$: set of rea-valued clocks,
- $D$: bounded integer variables,
- $I: l \rightarrow G(X)$: location invariant mapping,
- $E \subseteq L \times G(X) \times A \times 2^{\mathbb{D}(X)} \times L$ is a superset of directed edges:
  $I(x, \bar{l}) = (l, a, r, f) \in E$.

- Has Labeled Transition System (LTS) semantics.
- I/O, internal and timing non-determinism allow modelling parallelism, abstraction and possible time slacks.

- Test Spec: $(\{ E_1 \}|\{ E_2 \} \ldots |\{ E_n \}) \cup (\{ Z_1 \}|\{ Z_2 \} \ldots |\{ Z_n \}), A_{init}, A_{out}, T$.

---

Discriminating Power of Environments

- Consider traces: $0 \cdot Med!^1 \cdot High!^1 \cdot Med!^1 \cdot Low? \cdot r \ldots$

---

Randomized Test Generation and Execution Online

while $Z \neq \emptyset \land \text{iterations} \leq T$ do choose randomly:

1. if EnvOutput($Z$) $\neq \emptyset$ then // offer an input randomly choose $a \in$ EnvOutput($Z$) send $a$ to IUT $Z := Z \text{ After } a$.
2. randomly choose $\delta \in \text{Delays}(Z)$ // wait for an output sleep for $\delta$ time units and wake up on output $o$ if $o$ occurs at $\delta' \leq \delta$ then $Z := Z \text{ After } \delta'$ if $o \notin \text{ImpOutput}(Z)$ then return fail else $Z := Z \text{ After } o$
else $Z := Z \text{ After } \delta$ // no output within $\delta$ delay
3. $Z := \{(s_0, s_0)\}$, reset IUT //reset and restart if $Z = 0$ then return fail else return pass

---

Error Detection Capability: Mutant Experiment

- Specification: train-gate example of 9 timed automata.
- Implementation: 4 threads with a shared queue in C++.
- 7 mutants: M1-M6 with seeded error, M0 correct.

<table>
<thead>
<tr>
<th>Mutant</th>
<th>Number of input actions</th>
<th>Duration, min</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Avg</td>
</tr>
<tr>
<td>M1</td>
<td>2</td>
<td>4.8</td>
</tr>
<tr>
<td>M2</td>
<td>2</td>
<td>4.8</td>
</tr>
<tr>
<td>M3</td>
<td>2</td>
<td>4.7</td>
</tr>
<tr>
<td>M4</td>
<td>6</td>
<td>8.5</td>
</tr>
<tr>
<td>M5</td>
<td>4</td>
<td>5.6</td>
</tr>
<tr>
<td>M6</td>
<td>2</td>
<td>14.1</td>
</tr>
<tr>
<td>M0</td>
<td>3555</td>
<td>3751.4</td>
</tr>
</tbody>
</table>
**Benchmark Data: Summary**

<table>
<thead>
<tr>
<th>Mutant</th>
<th>Number of states in ( Z )</th>
<th>CPU execution time, ( \mu s )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>After (delay)</td>
<td>After (action)</td>
</tr>
<tr>
<td>M1</td>
<td>2.3  18  2.7  28</td>
<td>1113 3128 141 787</td>
</tr>
<tr>
<td>M2</td>
<td>2.3  22  2.8  30</td>
<td>1118 3311 147 791</td>
</tr>
<tr>
<td>M3</td>
<td>2.2  22  2.7  30</td>
<td>1112 3392 141 834</td>
</tr>
<tr>
<td>M4</td>
<td>2.8  24  3.1  48</td>
<td>1113 3469 125 936</td>
</tr>
<tr>
<td>M5</td>
<td>2.8  24  3.3  48</td>
<td>1131 3222 146 919</td>
</tr>
<tr>
<td>M6</td>
<td>2.7  27  2.9  36</td>
<td>1098 3531 110 861</td>
</tr>
<tr>
<td>M0</td>
<td>2.7  31  2.9  46</td>
<td>1085 3591 101 950</td>
</tr>
</tbody>
</table>

**Danfoss Case Study: EKC – Refrigeration Controller**

**Concluding Remarks**

- Online real-time testing theoretically **sound and complete in limit**.
- Environment assumptions should be **known and explicit**.
- Relativized conformance allows to **minimize cost of testing**.
- Implemented in TRON using **efficient algorithms from UPPAAL**.
- Encouraging **error detection capability and performance**.
- TRON allows **abstract and non-deterministic specifications**.
- **Extreme non-determinism may degrade performance**.
- Testable environments are limited by CPU and comm. latency.
- **IUT models just need to be deadlock free and input-enabled**.

**Future Work**

- **Research tasks:**
  - Clock synchronization, latency, jitter.
  - Coverage estimation, use coverage in guiding.
  - Diagnostics, fault location.
  - Model learning during experiment.
  - Relativized conformance in interface compatibility: unit testing.

- **Engineering tasks:**
  - New **UPPAAL features** (broadcast, committed, U-Code).
  - Termination of testing (specify property expressions?).
  - TRON in monitoring, testing via simulation and monitoring.
  - Relativized conformance in **practice**: specialized applications of generalized controllers, test-case guiding, debugging.
  - Industrial case studies.

**Performance Unpredictable: instances**

**Time Mapping to the Model and Back**

- Reachability algorithms for **afterDelay** and **afterAction**:
  
  \[
  \text{Closure}_{\text{after Delay}}(Z, d) = \bigcup_{(i, z) \in Z} \left\{ (i', z') : (i, z) \Rightarrow (i', z') \right\} 
  \]

  \[
  Z \text{ After } d = \left\{ (i, z') : (i, z) \in \text{Closure}_{\text{after Delay}}(Z, d), z' = (z, (t = d)) \right\}
  \]

- Everything above works well in controlled-time.
- But in real world, communication doesn’t happen instantaneously.
- Clocks at Env/tester and IUT may drift.
- Models of queues and drifts contain non-determinism.

**Summary and Future Work**

**Download...**

UPPAAL TRON is available for research and non-commercial use at:

http://www.cs.aau.dk/~marius/ton  

Thank You!