T-UPPAAL: Online Model-based Testing of Real-time Systems

Marius Mikucionis Kim G. Larsen Brian Nielsen
{marius,kgl,bnielsen}@cs.auc.dk
Department of Computer Science, Aalborg University, Fredrik Bajers Vej 7B, 9220 Aalborg Øst, Denmark

1. Introduction

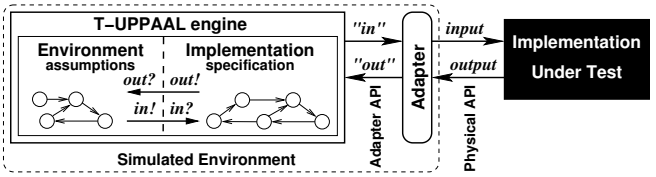
The goal of testing is to gain confidence in a physical computer based system by means of executing it. More than one third of typical project resources is spent on testing embedded and real-time systems, but still it remains ad-hoc, based on heuristics, and error-prone. Therefore systematic, theoretically well-founded and effective automated real-time testing techniques are of great practical value.

Testing conceptually consists of three activities: test case *generation*, test case *execution* and *verdict assignment*. We present T-UPPAAL—a new tool for model based testing of embedded real-time systems that automatically generates and executes tests "online" from a state machine model of the implementation under test (IUT) and its assumed environment which combined specify the required and allowed observable (real-time) behavior of the IUT. T-UPPAAL implements a sound and complete randomized testing algorithm, and uses a formally defined notion of correctness (relativized timed input/output conformance) to assign verdicts. Using *online* testing, events are generated and simultaneously executed.

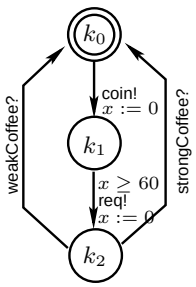
2. Modeling Embedded Systems

An embedded system interacts closely with its environment which typically consists of the controlled physical equipment (the plant) that is accessible via sensors and actuators, other computer based systems or digital devices accessible via communication networks using dedicated protocols, and human users. An embedded system must work correctly in this specific environment. The IUT and its environment communicate by exchanging $input$ and $output$ signals (seen from the IUT). When the IUT is being tested, the tester plays the role of the environment. The IUT is connected to T-UPPAAL via an adapter component that translates the abstract in/out actions into their real representation, and sends/receives them to/from the IUT, see Figure 1(a). Factoring in the explicit environment model allows T-UPPAAL to only generate realistic event sequences. Further, the user can exploit it to guide the T-UPPAAL to particularly interesting test situations.

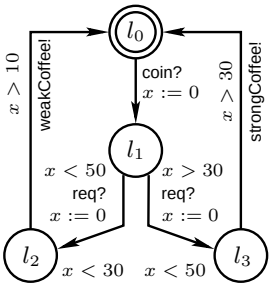
Timed Automata is an expressive and popular formalism for modeling real-time systems. Intuitively, a timed automaton is



(a) Automated testing of an embedded system.



(b) Example environment $\mathcal{E}_c$



(c) Example Specification $\mathcal{S}_c$.

Figure 1. Embedded system and example models.

an extended finite state machine equipped with a set of special real-valued variables called clocks whose value automatically increase as time progresses. Clocks may be used in enabling conditions on transitions and may be reset when a transition is executed. In addition, a clock invariant condition in a location limits how long the automaton may remain in that location. Figure 1(c) shows a timed automaton specifying the requirements to a coffee machine. It has a facility that allows the user, after paying, to indicate his eagerness to get coffee by pushing a request button on the machine forcing it to output coffee. However, allowing insufficient brewing time results in a weak coffee. Waiting less then 30 time units definitely results in weak coffee, and waiting more than 50 definitely in strong coffee.

Between 30 and 50 time units the choice is non-deterministic, meaning that the implementation/implementor may decide what to produce. After the request, it takes the machine an additional (non-deterministic) 10 to 30 (30 to 50) time units to produce weak coffee (strong coffee). The timed automata in Figure 1(b) models a potential (nice) user of the machine that pays before requesting coffee and that wants strong coffee thus requesting only after 60 time units.

3. Online Testing

Test cases can be generated from the model offline where the complete test scenarios and verdicts are computed a-priori and before execution. In contrast, *online testing* [1, 3, 2] combines test generation and execution: only a single test primitive is generated from the model at a time which is then immediately executed on the IUT. An observed test run is a timed trace consisting of an alternating sequence of (input or output) actions and time delays.

There are several advantages of online testing. First, testing may potentially continue for a long time (a single test run may take hours or even days), and therefore very long, intricate, and stressful test cases may be executed. Second, the state-space-explosion problem experienced by many offline test generation tools is reduced because only a very limited part of the state-space need to be stored at any point in time. Third, online test generators often allow more expressive specification languages, especially wrt. allowed non-determinism in real-time models, e.g., allowing timing uncertainty where an output event is expected in some interval of time, say between 1 and 5 time units from now. An online test generation algorithm is automatically adaptive to this non-determinism.

T-UPPAAL implements the algorithm shown in Algorithm 1. The main idea is to continually compute the set of states $\mathcal{Z}$ (consisting of state pairs (s, e) representing respectively the state of the specification- and the environment) that the specification can possibly occupy after the test run observed so far. The tester can perform three basic actions: either send an input (enabled output in the environment) to the IUT, wait for an output for some time, or reset the IUT and restart. If the tester observes an output or a time delay it checks whether this is legal according the state set. The state set is updated whenever an input is offered, or an output or delay is observed. Illegal occurrence or absence of an output is detected if the state set becomes empty which is the result if the observed trace is not in the specification. The function $\mathcal{Z}$ After α computes the set of states that can be reached after an observable action or after a time delay.

The algorithm manipulates large (even infinite) sets of states containing (real-valued) clock valuations. T-UPPAAL implements the required operations and analyses timed automata specification models using efficient symbolic algorithms based on solving clock inequations, so-called *zones*.

Algorithm 1 Initially $\mathcal{Z} := \{(s_0, e_o)\}$.

while $\mathcal{Z} \neq \emptyset \wedge \sharp iterations \leq T$ **do** randomly choose:
- *action*: `// offer an input`
  - **if** EnvOutput($\mathcal{Z}$) $\neq \emptyset$
    - randomly choose $a \in$ EnvOutput($\mathcal{Z}$)
    - send a to IUT
    - $\mathcal{Z} := \mathcal{Z}$ After a
- *delay*: `// wait for an output`
  - randomly choose $\delta \in$ Delays($\mathcal{Z}$)
  - sleep for δ time units and wake up on output o
  - **if** o occurs at $\delta' \leq \delta$ **then**
    - $\mathcal{Z} := \mathcal{Z}$ After δ'
    - **if** $o \notin$ ImpOutput($\mathcal{Z}$) **then return** *fail*
    - **else** $\mathcal{Z} := \mathcal{Z}$ After o
  - **else** `// no output within` δ `delay`
    - $\mathcal{Z} := \mathcal{Z}$ After δ
- *restart*: `//reset and restart`
  - $\mathcal{Z} := \{(s_0, e_0)\}$
  - **reset** IUT

if $\mathcal{Z} = \emptyset$ **then return** *fail*
else return *pass*

4. Implementation

UPPAAL is a mature (10 years;version 3.4.5), widely used, efficient model-checking tool for densely timed automata networks using symbolic reachability analysis. UPPAAL includes a graphical timed automata editor, a simulator, MSC visualization, and a verification engine. We have implemented T-UPPAAL by (non-trivially) extending the UPPAAL engine. T-UPPAAL currently runs on Solaris and Linux, with a Windows port being developed. Unlike UPPAAL, our implementation does not store the reached state space, but only the current symbolic state set. UPPAAL is available in various binary forms at [5], and a first official release T-UPPAAL at [4]. Both tools are free for non-commercial use.

References

[1] A. Belinfante, J. Feenstra, R. d. Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In 12^{th} *Int. Workshop on Testing of Communicating Systems*, pages 179–196, 1999.

[2] M. Krichen and S. Tripakis. Black-box Conformance Testing for Real-Time Systems. In *Model Checking Software: 11th International SPIN Workshop*, volume LNCS 2989. Springer, april 2004.

[3] M. Mikucionis, K. Larsen, and B. Nielsen. Online on-the-fly testing of real-time systems. Technical Report RS-03-49, Basic Research In Computer Science (BRICS), Dec. 2003.

[4] T-UPPAAL. W. S. www.cs.auc.dk/∼marius/tuppaal.

[5] UPPAAL. W. S. www.uppaal.com.