# Real-time System Testing On-the-fly[*]

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

**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 and still it remains ad-hoc, based on heuristics, and error-prone. Moreover, it is estimated that 99% of processors produced today are targeted for embedded applications. Real-time and embedded systems require a special attention to timing where the moment of input and output event appearance is as important as the event itself. Therefore a special attention must be paid to timing during testing. The goal of conformance testing is to check whether the behavior of the system under test (IUT) is correct (conforming) to that of its specification. We follow a model driven approach where a formal model (or specification) defines the required (real-time) observable behavior of the IUT, and from this we automatically derive and execute real-time test cases to determine whether the IUT is conforming.

A new approach to model based test generation is (on-line) on-the-fly testing that 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 implementation. Then the output produced by implementation is checked against the specification, and new test primitive is produced, and so forth until it is decided to end the test. An observed test run is a timed trace consisting of an alternating sequence of (input or output) actions and time delays. The main advantages of on-the-fly testing is that very long and stressful test cases are executed, and that the state-space-explosion problem is reduced during test generation, because only a limited part of the state-space need to be stored. On-the-fly testing from Promela [3] and Lotos specifications for un-timed systems have been implemented in the TorX [2] tool, and practical application to real case studies show promising results [2]. However, TorX provides no support for real-time systems. We present the framework, the algorithm and the tool for on-the-fly testing of real-time systems based on UppAal. UppAal is a timed automata model checker developed jointly by a group of researches at Uppsala University and Aalborg University. We adopt the UppAal model specification language and extend the verification engine for the on-the-fly testing. The most important feature of UppAal is the use the symbolic techniques to handle real-valued clocks of timed automata.

**Testing framework.** The test framework consists of the implementation under test (IUT) and its environment that is to be simulated by the tester. In our case the tester is a test computer equipped with a *test specification*, an adopted UppAal engine and an *adapter*, see Figure 1. An IUT usually

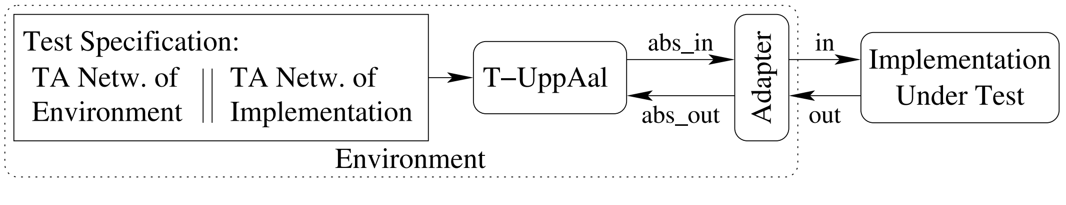

Figure 1: On-the-fly testing framework.

operates in a particular environment - a collection of conditions and assumptions that the IUT is used under. Not every environment is realistic or reasonable and therefore only tests relevant to this

[*]Funded by Basic Research in Computer Science.

environment should be executed. Moreover we may want to test how system would behave under very specific conditions, e.g. try some test purpose which led to a failure. Therefore the test specification is allowed to be a parallel composition of a model of the implementation and a model of environment. The adapter component translates the abstract input and output representation (*abs_in*, *abs_out*) into real actions (*in*, *out*) applied to and received from the IUT. We assume that the IUT and its model are input enabled to be able to accept any input offered at any moment in time. The tester is allowed to offer any input allowed by the environment from the test specification.

**Conformance Relation.** A conformance relation defines what IUT behaviors are considered correct compared to its specification. SPIN [3] uses the non-timed conformance relation *ioco*. Similarly we propose *timed trace inclusion* as conformance relation meaning that the timed traces of the IUT must be included in the timed traces of specification. The conformance relation forms basis for a test verdict - *pass*, *fail* or *inconclusive*. The intuition of a verdict is as follows:

- *pass* is given if all the observed outputs (including passage of time) were allowed by the specification. The passage of time implies that the IUT may stay silent (not produce an output) only if allowed so by the specification;
- *fail* is given if some output observed was not permitted by the model of implementation;
- *inconclusive* is given if some output observed was not expected by the environment model (an unpredicted event which prevented to reach the goal of the test).

The maximum duration of the test is defined in the test specification as a time-out for the testing. We consider clocks to be of real-valued density. Therefore we reuse the *symbolic techniques* [6] implemented in UPPAAL, which allows us to analyze continuum sets of clock values in compact data structures.

**Symbolic on-the-fly test algorithm.** The test generation and execution algorithm is based on maintaining the current reachable symbolic state set $Z$ representing all states that the test specification can possibly occupy after the timed trace observed so far. Knowing this allows us to choose appropriate test primitives and to validate IUT outputs. Initially $Z$ contains a single symbolic state $\langle \bar{l}_0, \bar{0} \rangle$ where $\bar{l}_0$ is the initial location vector of timed automata network and $\bar{0}$ is the initial time zone where all clocks are set to zero. See the details in Algorithm 1.

**Algorithm 1** Test generation and execution. Initially $Z := \{\langle \bar{l}_0, \bar{0} \rangle\}$.

```
while $timeout > 0$ do choose (randomly) one of the following two:
    $action$:          // offer an input
        $a := ChooseAction(EnvOutput(Z))$
        send $a$ to implementation
        $Z := After(Z, a)$
    $delay$:           // wait for an output
        $\delta := ChooseDelay(Z)$
        sleep for $\delta$ time units and wake up on output $o$
        if $o$ occurs at $\delta' \leq \delta$ then
            $Z := After(Z, \delta')$
            if $o \notin ImpOutput(Z)$ then return fail
            else if $o \notin EnvInput(Z)$ then return inconclusive
            else $Z := After(Z, o)$
            $timeout := timeout - \delta'$
        else           // no output occurred within $\delta$ time
            $Z := After(Z, \delta)$
            if $Z = \emptyset$ then return fail
            $timeout := timeout - \delta$
return pass
```

The functions used in Algorithm 1 query the UPPAAL engine for the return values:

- *After* computes a closure of symbolic states after all potential environment and IUT internal actions and returns the next set of reachable symbolic states that can be reached after performing an input or output test event[1] or a delay. Due to non-determinism in the specification this requires a representation as set of symbolic states as opposed to a single state and single symbolic state, and the underlying algorithms for computing it are un-trivial, and cannot be presented due to space limitations. *After* returns an empty set if the delay or action was not allowed by the specification.

- *EnvInput*, *EnvOutput* and *ImpOutput* compute the applicable input and output actions for the model of respectively the environment and the model of implementation;
- *ChooseDelay* and *ChooseAction* selects a delay and an action applicable to IUT when it is in some state mentioned in $Z$. The *ChooseAction* and *ChooseDelay* functions currently selects actions or delays at random.

Different strategies can be applied to guide the test generation to "interesting" or uncovered states by changing the model of environment, "choose" functions and adopting them to a particular test purposes. The best results are expected in symbiosis with [5], [7] and other future works in this area.

**Status and future work.** The current work on the on-the-fly test generator resulted in the first prototype of T-UPPAAL (Testing-UPPAAL) [1], which showed promising results in generating test from simple system models, such as train-gate controller [4]. The concept is realizable, and functional, and the performance of the symbolic state set computation algorithms appear fast enough for many realistic real-time systems. However, very large specification with an extreme amount of non-deterministic behavior may need significant time to compute the reachable symbolic state set, and may need improved algorithms.

The future T-UPPAAL prototype development includes enhanced test generation and execution algorithm with further test event selection strategies and improved clock synchronization between the tool and IUT. The *data value passing* protocol described in [1] must be implemented to enable the data value communication between the tool and the IUT. Also we plan to apply it to realistic embedded and real-time systems and protocols.

## References

[1] Marius Mikucionis and Egle Sasnauskaite. On-the-fly Testing Using UPPAAL. Aalborg University. Distributed Systems and Semantics unit, Department of Computer Science. Master thesis, June 2003. URL: http://www.cs.auc.dk/∼marius/master.pdf

[2] R. de Vries, J. Tretmans, A. Belinfante, J. Feenstra, L. Feijs, S. Mauw, N. Goga, L. Heerink, and A. de Heer. Côte de Resyste in PROGRESS. In STW Technology Foundation, editor, PROGRESS 2000 - Workshop on Embedded Systems, pages 141-148, Utrecht, The Netherlands, October 13 2000.

[3] Rene de Vries, Jan Tretmans. On-the-Fly Conformance Testing Using Spin. University of Twente. Formal Methods and Tools group, Department of Computer Science. P.O. Box 217, 7500 AE Enschede, The Netherlands.

[4] Wang Yi, Paul Pettersson and Mats Daniels. Automatic Verification of Real-Time Communicating Systems by Constraint Solving. In Proceedings of the 7th International Conference on Formal Description Techniques, pages 223-238, North-Holland. 1994.

[5] Anders Hessel, Kim G.Larsen, Brian Nielsen, Paul Pettersson, Arne Skou. Time-optimal Real-Time Test Case Generation using UPPAAL. To appear in the 3rd International Workshop on Formal Approaches to Testing of Software (FATES'03). Montreal, Canada,

[6] Gerd Behrmann, Johan Bengtsson, Alexandre David, Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal Implementation Secrets. In Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'02), 2002. URL: http://www.uppaal.com/

[7] Brian Nielsen and Arne Skou. Automated Test Generation from Timed Automata. In proc. TACAS 2001 - Tools and Algorithms for the Construction and Analysis of Systems (Tiziana Margaria and Wang Yi Eds.), 343–357, Genova, Italy, April 2–6, 2001.

[1] We call the observable channel synchronization a an *action*, since the action may carry more information than a channel synchronization alone.