UPPAAL*: Status & Developments

Kim G Larsen[1] *Paul Pettersson*[2] *Wang Yi*[2]

[1] Department of Computer Science and Mathematics, Aalborg University, Denmark.
[2] Department of Computer Systems, Uppsala University, Sweden.

1 Introduction

UPPAAL[3] is a tool box for validation (via graphical simulation) and verification (via automatic model-checking) of real-time systems, based on constraint solving and on-the-fly techniques. It consists of three main parts: a description language, a simulator and a model-checker. It is appropriate for systems that can be modelled as networks of timed automata [3, 2], i.e. a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels and shared variables. The description language of UPPAAL is a non-deterministic guarded command language with data types (currently, only `integer` and `clock`, with restricted forms of operations implemented). The semantics of the language is given in terms of labelled transition systems in the tradition of timed process algebras. The simulator enables examination of *possible* dynamic executions in early design stages and thus provides an inexpensive mean of fault detection prior to verification by the model-checker which covers the exhaustive dynamic behaviour.

The two main design criteria for UPPAAL have been *efficiency* and *ease of usage*. An important key to the efficiency of the current model-checking engine of UPPAAL is the application of *on-the-fly* verification combined with a *symbolic* technique reducing the verification problem to that of solving simple *constraint systems* [3, 2]. In contrast to the previous version of UPPAAL which was based on backwards reachability analysis, the current version implements forwards on-the-fly reachability analysis. In addition, it offers both breadth-first and depth-first search of the state-space of a system description. Another important key to efficiency is the restriction to model checking of simple invariant and reachability properties. Other properties such as bounded liveness properties may be checked by reasoning about the system in the context of a testing automata or the decorated system with debugging information. In order to facilitate debugging, UPPAAL automatically generates a *diagnostic trace* that explains why a property is (or is not) satisfied by a system description. Our current research results promises even more efficient verification engines in near future (see Section 3).

To ease the usage of UPPAAL particular effort has been made in developing graphical *user interfaces*. Thus, system descriptions may be defined graphically using an

---

* UPPAAL is developed in collaboration between the Department of Computer Systems at Uppsala University (UPP), Sweden and BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation) at Aalborg University (AAL), Denmark. The people involved with the development are Wang Yi (UPP), Kim G. Larsen (AAL), Paul Pettersson (UPP), Johan Bengtsson (UPP), Fredrik Larsson (UPP), Kåre J. Kristoffersen (AAL), Palle Christensen (AAL), Jesper Gravgaard (AAL), Per S. Jensen (AAL), and Thomas M. Sørensen (AAL).

[3] Installation and documentation available at `http://www.docs.uu.se/docs/rtmv/uppaal/`.

AUTOGRAPH-based interface which is automatically transformed into textual format[4]. Also (certain) multi-rate timed automata are transformable into timed automata.

2 From Verification to Validation

The main novelty in the new UPPAAL version is the addition of a *graphical simulator*, which enables visualisation and recording of possible dynamic behaviours of a system descriptions in terms of sequences of symbolic states of the system. Reports from a number of UPPAAL users indicate that the addition of the simulator significantly enhances the tool box as it allows for inexpensive fault detection in the very early modelling stages. The simulator enables the user to perform *informal validations* and thus to obtain a better understanding of the dynamic (mis)behaviour of a system complementing the existing *formal verification* engine of UPPAAL. In particular, the diagnostic traces generated by the verifier in case of an erroneous system may be graphically visualised using the simulator. During a simulation the following information is presented to the user:

- The current symbolic state. It consists of the *discrete location vector* visualised by highlighted location markings directly in the graphical description and the *constraints on the clocks and data variables* displayed in a separate window.
- The possible next transitions. They are displayed and may be selected; simultaneously the corresponding edges are highlighted in the graphical description of the timed automata.
- The trace that leads to the current symbolic state. It is displayed and may be saved, reexamined, replayed and reset from any intermediate point. In particular diagnostic traces generated by the verifier may be loaded for examination.

The simulator may run either in an interactive mode, where the user selects transitions, or in an automatic mode, where the simulator itself randomly selects transitions. During a simulation various parts of the information may be hidden, e.g. the constraints over clocks and integer variables in the current state. Also, display of the generated trace may be omitted.

3 New Developments

The basic model employed by UPPAAL is that of networks of timed automata extended with data variables. To meet requirements arising from various case studies this basic model has been extended to include features such as *Committed Locations* and *Urgent Channels*.

Various techniques for optimising the space- and time-performance of the reachability engine of UPPAAL has been developed and experimentally examined. The following techniques has already been added to the current UPPAAL version:

Re-Use. This idea is extremely simple, but yields nevertheless significant improvements in time-consumption. Whenever a system is analysed with respect to *several* reachability properties the computed portion of the reachable (symbolic) state-space is re-used, hence avoiding time-consuming recomputations.

[4] Of course systems may also be defined using the textual notation directly.

Control Structure Analysis. During a standard reachability analysis all new encountered symbolic states are normally stored in a global data structure (called the `Passed` list) in order to ensure termination. However, this is not needed if a predecessor of the new symbolic state is already presented in the `Passed` list. In fact, to ensure termination, it suffice to save only one state for each dynamic loop. An improved on-the-fly reachability algorihtm based on this stragety has been implemented and demonstrates significant space-savings (see Table 1).

The following techniques have been experimentally examined and show high potentials:

Compact Data Structures for Constraints. In the present implementation of UPPAAL the constraint part of a symbolic state is represented using the well known Difference Bounded Matrices (DBM) allowing for efficient emptiness and inclusion checks as well as efficient transformations of the constraints according to the dynamic behaviour of the timed system. The DBM representation gives an explicit bound for the difference between each pair of clocks (and each individual clock), hence using space corresponding to $(n+1) \times (n+1)$ integers. However, in practice it often turns out that most of these bounds are redundant. Recently, we have developed an $\mathcal{O}(n^3)$ algorithm that given a DBM constructs a canonical and minimal set of constraints representing the same solution set[5]. Thus, when saving a symbolic state in the `Passed` list, we use the (often substantially smaller) canonical reduction of the constraint system. Experimental results demonstrates truly significant space-savings (see statistics in Table 1) and the technique will in short time be part of the official UPPAAL distribution.

Compositional Verification. In a number of papers (e.g. [2]) we have developed the theoretical basis for a compositional verification method which allows components of a network of timed automata to be gradually moved from the system description into the specified property, thus avoiding global state-space considerations. Recently, a prototype implementation in C++ has been made giving experimental evidence of the potential of the technique [1]. Using only 172.3 seconds and 32MB main memory the tool automatically verifies Fischer's protocol with 50 processes.

Table 1 compares the Control Structure Reduction (CSR), the Compact Data Structures for Constraints technique (CDSC), and their combination (CDSC & CSR) with the performance of the standard reachability algorithm of UPPAAL (Standard). Space is measured in number of clock constraints and time is measured in seconds. The improvement of space-performance observed is remarkable.

4 Applications

Since its first release in 1995, UPPAAL has been applied in a number of case-studies by users from both academic and industrial sites. They can be roughly divided in two classes: real-time controllers and communication protocols.

Real-Time Controllers: The representative example in this category is the design and analysis of a gear-box controller for vehicles using UPPAAL by Mecel AB[6]. Other known

[5] Given a weighted, directed graph with n vertices, the algorithm constructs in time $\mathcal{O}(n^3)$ a reduced graph with the minimal number of edges having the same shortest path closure as the original graph.

[6] Mecel AB is a Swedish company developing control systems for vehicle industries.

	Standard		CDSC		CSR		CDSC & CSR	
	space	time	space	time	space	time	space	time
Audio	828	0.73	219	0.60	774	0.61	206	0.63
Audio w. Coll.	1 902 816	1 839.21	443 221	1 376.09	886 284	1 343.41	205 734	1 104.76
Box Sorter	625	0.40	139	0.36	150	0.41	32	0.41
Fischer 2	225	0.41	44	0.35	99	0.37	16	0.35
Fischer 3	3 376	0.71	621	0.68	1 376	0.70	237	0.64
Fischer 4	56 825	11.58	9 352	9.95	22 400	9.56	3 528	8.94
Fischer 5	1 082 916	789.15	158 875	651.61	419 112	650.00	59 715	605.23
M. Plant	96 084	14.36	27 042	18.03	51 048	16.44	14 968	16.85
Train Gate	432	0.46	130	0.39	384	0.40	114	0.42

Table 1. Performance Statistics.

examples are the steam generator, the train gate controller, the manufacturing plant, and the mine-pump controller.

Real-Time Communication Protocols: UPPAAL has been mainly applied to model and verify protocols where correct timing is critical, including the bounded retransmission protocol, the collision avoidance protocol, and the audio-control protocol designed by Philips.

In terms of complexity, Philips audio-control protocol with bus-collision is the most comprehensive case-study so far where UPPAAL has been applied. The protocol was developed by Philips to exchange information between components in one of their high-end audio sets. The version of the protocol with *bus-collision handling* was verified using the previous version of UPPAAL installed on a SGI ONYX machine. The main correctness property consumed 7.5 hours and 527.4 MB of memory. Using the current version of the UPPAAL verifier the same property may be verified in 15.49 minutes and 31 MB of memory on a Sun Sparc 4.

References

1. Kåre J. Kristoffersen, Francois Larroussinie, Kim G. Larsen, Paul Pettersson, and Wang Yi. A compositional proof of a real-time mutual exclusion protocol. In *Proc. of the 7*th *International Joint Conference on the Theory and Practice of Software Development*, April 1997.
2. Kim G. Larsen, Paul Pettersson, and Wang Yi. Model-Checking for Real-Time Systems. In *Proc. of Fundamentals of Computation Theory*, volume 965 of *Lecture Notes in Computer Science*, pages 62–88, August 1995.
3. Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-Time Communicating Systems By Constraint-Solving. In *Proc. of the 7*th *International Conference on Formal Description Techniques*, 1994.

This article was processed using the LaTeX macro package with LLNCS style