UPPAAL 4.0

Gerd Behrmann
and Alexandre David
and Kim G. Larsen
Department of Computer Science
Aalborg University, Denmark
Email: {behrmann,adavid,kgl}@cs.aau.dk

John Häkansson
and Paul Pettersson
and Wang Yi
Department of Information Technology
Uppsala University, Sweden
Email: {johnh,paupet,yi}@it.uu.se

Martijn Hendriks
Institute for Computing
and Information Sciences
Radboud University Nijmegen
The Netherlands
Email: m.hendriks@cs.ru.nl

Abstract—UPPAAL 4.0 is the result of over two and a half years of development and contains many new features, additions to the
modeling language, performance improvements, enhancements and polish to the the easy to use graphical user interface, and
is accompanied by several open source libraries. The tool and
libraries are available free of charge for academic, educational
We describe three of the new features: User defined functions,
priorities and symmetry reduction.

I. INTRODUCTION

UPPAAL is a verification tool for timed automata. Its focus
on speed and usability has made the tool popular both as
a teaching tool in academia, as a gentle introduction to the
world of model checking, and as a tool for doing serious
case studies as witnessed by the large number of publications
in which UPPAAL was used. Version 4.0 was released in
May 2006 and introduces many new features that increase
the applicability and the performance of the tool. This paper
describes three major features of the new release. Many more
are worth describing (such as a typical reduction in memory
usage of a factor 3 to 5, new abstraction techniques resulting
in large performance increases [1], or our implementation of
the generalised sweep line method), however lack of space
forces us to focus on the most visible changes.

II. USER DEFINED FUNCTIONS

Many problems require non-trivial computations with com-
plex control-flow to be embedded in the model. In a graphical
language like the one used by UPPAAL, this tends to clutter
the model and makes the model hard to read and maintain.
Often, including such computations in the model enlarges the
state space by introducing intermediate states and irrelevant
interleaving. In the past, this has led to the addition of
committed locations in UPPAAL to build atomic sequences.

In UPPAAL 4.0 we have extended the modeling language
with user defined functions. These are fully integrated into the
modeling language, and have access and can modify all state
variables. The syntax follows the style of C/C++/Java, and
most control-flow constructs of C are supported. Functions are
evaluated atomically and must be deterministic, whereby inter-
mediate states are avoided (similar to the d_step construction
in SPIN, which marks a sequence of statements as atomic and
deterministic). The only limitations are that recursive calls
are not allowed and functions must eventually return. The
second requirement is currently not enforced by UPPAAL, and
UPPAAL will not terminate if a user defined function enters
an infinite loop (this is a technicality and the language was
designed such that future versions of UPPAAL can be extended
to analyze the model for such problems).

User defined functions are compiled to byte-code, and
executed at verification time on a small embedded stack
machine. We have decided not to use an external compiler
like gcc, even though this would result in faster evaluation of
complicated functions. The dependence on an external comp-
iler would complicate the installation procedure of UPPAAL
considerably. We do not expect the virtual machine to be a
bottleneck for verification, as most time is spent on other
operations anyway.

The extension has proven to be extremely useful in almost
all UPPAAL models. Typical uses include that of naming or
hiding complicated expressions from the graphical language,
performing updates on data structures like routing tables,
queues, and stacks (although, the size of such data structures
must always be bounded), and performing computations that
require loops and complex control-flow. One of the more
ingenious uses we have seen, is to implement an interpreter
for live sequence charts, which when embedded in a UPPAAL
model can be used to check conformance.

III. PRIORITIES

In the implementation of real-time systems, the priorities
are often associated with processes (or tasks) to structure
and control the usage of shared resources such as CPU or
shared memory. As a consequence, programming languages
and scheduling policies used in real-time operating systems are
often based on a notion of priorities on tasks. In lower levels,
priorities are often associated with interrupts to hardware
devices and access to e.g., shared communication buses.

In UPPAAL 4.0 we have extended the modelling language
with priorities on channels and automata [2]. The priority
orders defined in the model are translated into a priority order
on internal and synchronizing transitions. At a given time-
point, an enabled transition will block (disable) another if it
has a higher priority.

The example below specifies that channel a has a lower
priority than channels b and c, and that the automaton P has
a lower priority than automata $Q$ and $R$. In such a model, with priorities on both automata and channels, we resolve priorities by comparing priorities on channels first. If they are the same, the automata priorities are compared. For efficient model-checking the priority orders are total orders.

chan priority a < b, c;
 system P < Q, R;

**UPPAAL** uses difference bound matrices (DBMs) [3] to represent convex constraints on clock variables. The efficient implementation of priorities is made possible by the introduction of a optimised subtraction operation in the DBM library. However, the result of the subtraction is not necessarily a convex zone, but rather a set of zones. The algorithm we use for subtraction generates a disjoint set of DBMs, and uses a heuristic to minimise the number of DBMs in the result. The library even supports merging zones back together.

### IV. Symmetry Reduction

Symmetry reduction is a well-known technique to alleviate the state-space explosion problem and has recently been added to **UPPAAL**. This technique can be applied to models that contain multiple equivalently behaving processes [4]. Consider, for instance, Fischer’s mutual exclusion algorithm. It consists of a set of processes that only differ in their “identity”. A process writes its identity in a global variable and checks after a while whether this global variable still contains its identity. If so, it can enter the critical section.

In order to provide convenient and safe usage of symmetry reduction, the modeling language of **UPPAAL** has been extended with the *scalar* datatype. It can be used to define *scalarsets* which can be regarded as unordered integer subranges. As a result, the only operations allowed on scalars are (i) equivalency testing with a scalar of the same type and (ii) assignment with a scalar of the same type. Furthermore, scalars can be used to index arrays if the dimension of the array has the same type as the indexing scalar. In the model for Fischer’s mutex algorithm, the process identity is a scalar and therefore there is a single scalarset.

Symmetry reduction can – in theory – be very beneficial: its savings w.r.t. the size of the reachable state space can (for a model with one scalarset of size $n$) approach a factor $n!$. This super-exponential gain is, unfortunately, not always observed due to a fundamental technical issue. Fig. 1 shows the effect of symmetry reduction for Fischer’s mutex algorithm.

An important application area of symmetry reduction is the area of distributed algorithms since models of instances of such algorithms often consist of a number of symmetric processes. For instance, symmetry reduction in **UPPAAL** has been succesfully applied to the leader election algorithm in [5], a consensus algorithm [6] and the Zeroconf protocol [7].

![Fig. 1. The effects of symmetry reduction on the time and memory consumption of UPPAAL 4.0 for Fischer’s mutual exclusion algorithm.](image)

### V. Conclusion

We have described three of the most visible new features of **UPPAAL** 4.0. The DBM library is released under the GPL and contains language bindings for C, C++ and Ruby. The parser library is released under the LGPL and is ideal for implementing model transformation tools or analysis tools. Java parsers and client stubs for the verification backend are distributed with **UPPAAL** and may be used by, e.g., domain specific tools (the documentation can be found at our web site). Finally, **UPPAAL** can be used as a compiler for **UPPAAL** models, translating the model to a byte-code representation.

### ACKNOWLEDGMENT

The authors thank Kim Algreen, Marius Mikucionis, Leonid Mokrushin, Ulrik Nyman, Robert Olesen, and Jacob Illum Rasmussen for their contributions to **UPPAAL** 4.0.

### REFERENCES


