Introduction to UPPAAL.

Stefaan Kenis

Department of cumputer science
Universiteit Antwerpen
stefaan.kenis@student.uantwerpen.be

Abstract

UPPAAL is a tool for modelling, simulation and verification of Real-Time systems that is developed at the universities of Aalborg and Uppsala. It has both a graphical and a textual interface for describing timed automata. This tool is mostly used for systems that require communication and timing aspects. The two main design criteria for this tool are efficiency and ease of usage. The main properties that can be tested with UPPAAL are safety and bounded liveness.
The current version of UPPAAL is available on `http://www.uppaal.org`.

Keywords: real-time systems, timed automata, safety, bounded liveness, modelling, simulation, verification, reachability analysis

1. Introduction

In this paper we will describe UPPAAL, its features, examples, performance and future work. UPPAAL is mostly used for real-time systems, but it can be used for all systems that can be described by (timed) automata. Section 2 will give an short overview of UPPAAL (Bengtsson et al., 1995, 1996; Larsen et al., 1997a). Section 3 will explain the features of UPPAAL (Larsen et al., 1997b,a; Amnell et al., 2001; Behrmann et al., 2001, 2004). Section 4 will be about the properties that can be tested with UPPAAL (Larsen et al., 1997b; Amnell et al., 2001; Behrmann et al., 2004). Section 5 will give some example cases (Bengtsson et al., 1995, 1996; Larsen et al., 1997b,a). In section 6 the performance is analysed (Bengtsson et al., 1995; Amnell et al., 2001; Behrmann et al., 2001). Section 7 is about future work.

2. Overview of UPPAAL

UPPAAL is a tool that is developed by the universities of Aalborg and Uppsala in 1995. It is a tool for modelling, simulating and verifying real-time systems. The next image explains the structure of the tool.

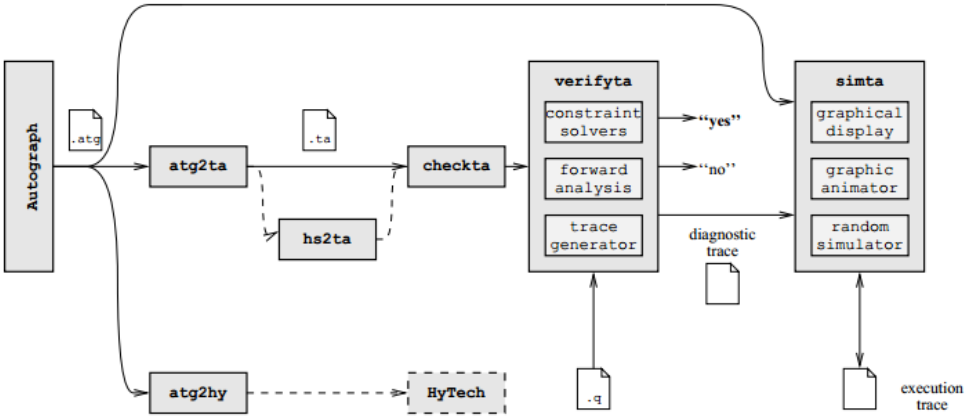


Figure 1: Overview of UPPAAL.

In this tool we can specify our models both graphical and textual. The first part of this image represents the transformation of an .atg file to an .at file. This a transformation from a graphical format to a textual format. The hs2ta box represents a transformation from linear hybrid automata to several simple timed automata. Hybrid automata can have clocks with rates between a certain interval. Then a few simple syntactical checks are done in the checkta box. If all this is done we can verify our model for a certain property. This gives yes or no as an answer and it also gives a diagnostic trace. This trace explains why the answer is yes or no and can be very useful when the answer is not what is expected. We can also simulate graphical models.

3. Features

The tool consists of three parts: an editor, a simulator and a verifier. With the editor models can be created or modified.

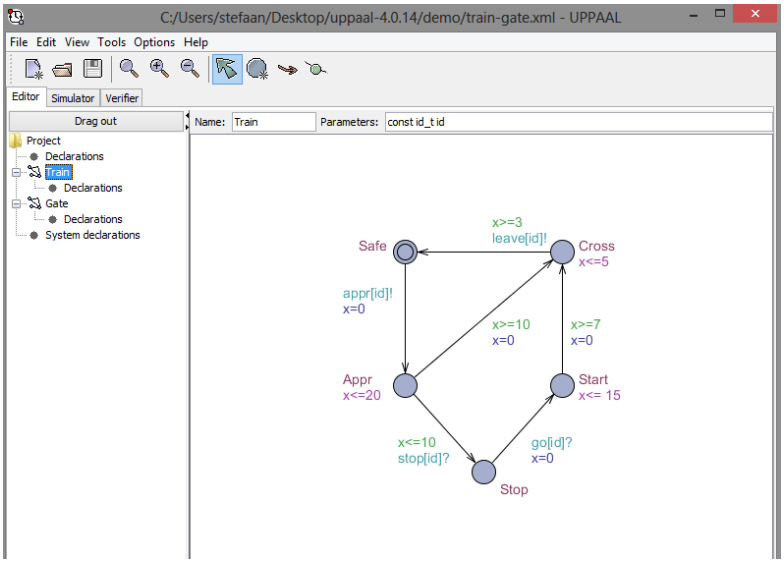


Figure 2: The Editor.

These models can be simulated with the simulator. At each point it consists of the current symbolic state, possible transitions to the next state and a trace that leads to the current state. Each step can be done interactively by the user or automatically. All steps and traces are kept and can be saved and loaded. This way the user can have a very fast way of fault detection. It also can be used to execute a trace that was given by the verifier.

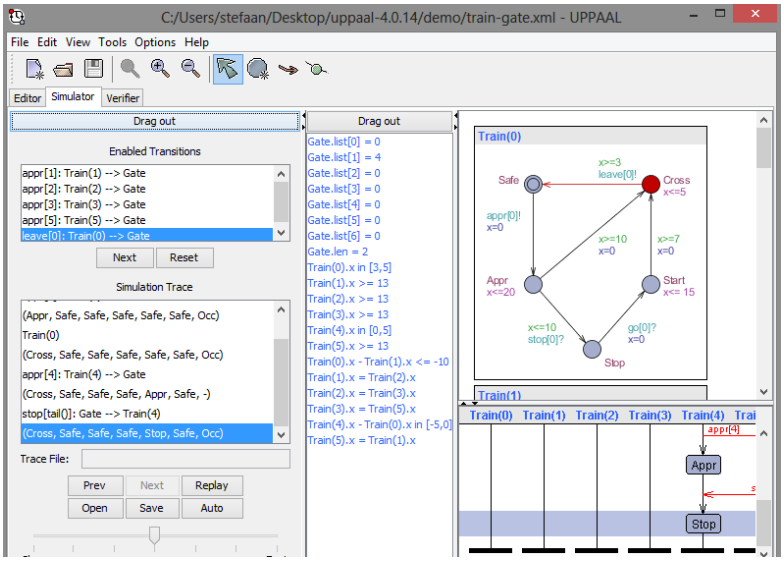


Figure 3: The Simulator.

The verifier is used to formally check some properties.

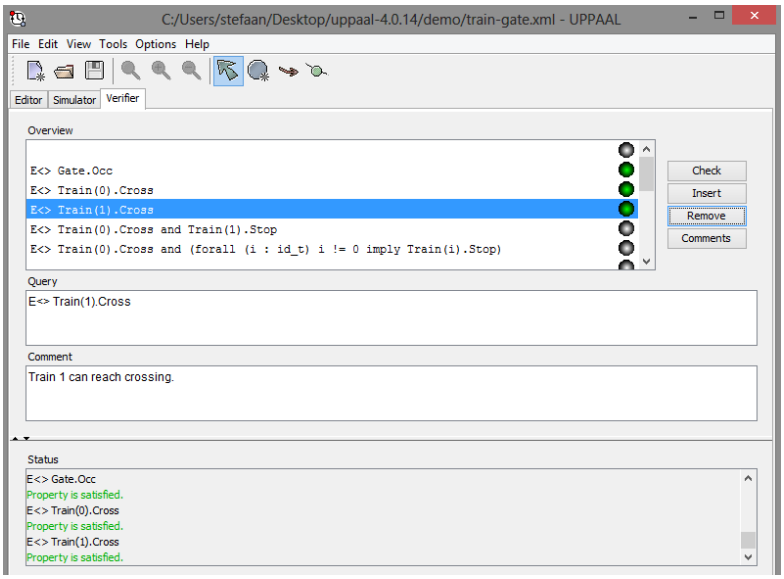


Figure 4: The Verifier.

This tool also supports data variables, constants, arrays, guards, clocks,...

4. Properties

In UPPAAL it is possible to specify invariants and to do reachability analysis. This reachability analysis can be done with both Forward and Backward reachability analysis and both can make use of bread-first or depth-first search. The two main properties that can be checked are safety and bounded liveness. With the safety property it is possible to check if certain 'bad' state can never occur. With the bounded liveness we can check if a certain 'good' state will eventually occur. If the result is not what was expected the diagnostic trace can be used for debugging.

5. Example cases

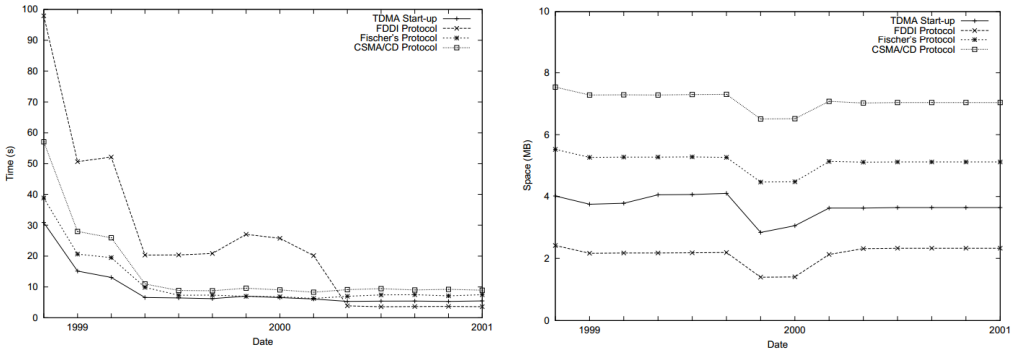
When UPPAAL is installed there are some example cases created:

- Two doors: Two doors that can not be open at the same time and a door needs a few seconds to open, stay open, close and stay closed.
- Bridge: Four vikings that need to cross a damaged bridge in a certain time. The bridge can only carry two vikings at the same time and each viking needs a certain amount of minutes to cross the bridge.
- Fischer: Fischer's mutual exclusion protocol.
- Train-gate: A bridge as a shared resource and several trains that can pass the bridge one at the time.
- ...

All this example are similar in the fact that they all should satisfy some safety and bounded liveness properties. All this example cases also have timing aspects.

6. Performance

One of the main criteria for UPPAAL was efficiency. This means that the developers wanted a tool that is efficient in both space and time. They did a lot of improvements for this over the years. This can be seen by the following image:



(a) Time (b) Space

Figure 5: Time and space improvements over the years.

They also claim that for all their examples their tool was faster than other tools and that it could handle larger systems. This is presented in the following image:

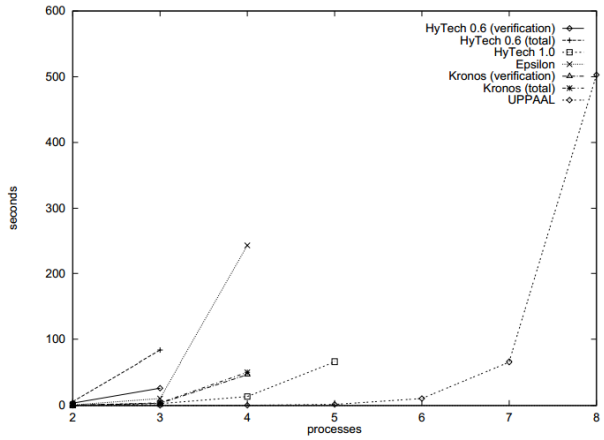


Figure 6: Execution times of Fischer's mutual exclusion protocol.

The tool does not have state-space explosion problems, because they use on-the-fly searching techniques that makes use of solving simple linear constraints. Another advantage of this tool is that is reuses parts of the reachable state-space when several properties are checked. In the newer versions they have also included the possibility for distributed state-space exploration.

7. Future work

In the near future I will use UPPAAL to analyse role-playing games. I will start by creating RPG models in UPPAAL with the editor. Then I will simulate and verify them on both safety and bounded liveness. For example the deadlock states are interesting, because we only want a deadlock when the game finishes. The models are saved in an xml-format, so it could be possible to export models from AToMPM to metaDepth and then transform it into a valid xml-file that is used by UPPAAL.

References

Amnell, T., Behrmann, G., Bengtsson, J., D'Argenio, P.R., David, A., Fehnker, A., Hune, T., Jeannet, B., Larsen, K.G., Möller, M.O., Pettersson, P., Weise, C., Yi, W., 2001. UPPAAL - Now, Next, and Future, in: Cassez, F., Jard, C., Rozoy, B., Ryan, M. (Eds.), Modelling and Verification of Parallel Processes, Springer–Verlag. pp. 100–125.

Behrmann, G., David, A., Larsen, K.G., 2004. A tutorial on UPPAAL, in: Bernardo, M., Corradini, F. (Eds.), Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004, Springer–Verlag. pp. 200–236.

Behrmann, G., David, A., Larsen, K.G., Mller, O., Pettersson, P., Yi, W., 2001. UPPAAL - present and future, in: Proc. of 40*th* IEEE Conference on Decision and Control, IEEE Computer Society Press.

Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W., 1995. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems, in: Proc. of Workshop on Verification and Control of Hybrid Systems III, Springer–Verlag. pp. 232–243.

Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W., 1996. UPPAAL in 1995, in: Proc. of the 2*nd* Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Springer–Verlag. pp. 431–434.

Larsen, K.G., Pettersson, P., Yi, W., 1997a. UPPAAL in a Nutshell. Int. Journal on Software Tools for Technology Transfer 1, 134–152.

Larsen, K.G., Pettersson, P., Yi, W., 1997b. UPPAAL: Status and developments, Springer–Verlag. pp. 456–459.