Uppaal-Tiga: Timed Games for Everyone

Gerd Behrmann[1], Agnès Cougnard[1], Alexandre David[1], Emmanuel Fleury[2],
Kim G. Larsen[1], Didier Lime[3]

[1] CISS, Aalborg University, Aalborg, Denmark
{behrmann,acougnar,adavid,kgl}@cs.aau.dk

[2] LaBRI, Bordeaux-1 University, CNRS (UMR 5800), Talence, France
fleury@labri.fr

[3] IRCCyN, École Centrale de Nantes, CNRS (UMR 6597), Nantes, France
Didier.Lime@irccyn.ec-nantes.fr

Abstract. In 2005 we proposed the first efficient on-the-fly algorithm for solving games based on timed game automata with respect to reachability and safety properties. Since then we have completely re-implemented the first prototype and made dramatic improvements both in terms of performance (several orders of magnitude) and the input language (complete support of all the language features of UPPAAL). In addition, the tool supports the new feature of strategy generation with different compactness levels. In this paper we present this new version of UPPAAL-TIGA.

1 Introduction

UPPAAL-TIGA[1] is a tool for solving games based on timed game automata. It is implementing the on-the-fly algorithm presented in [CDF+05]. The new version is faster by several orders of magnitude (more than 1000 times faster on large examples) and consumes much less memory (100 times less on large examples) than the first prototype presented at CONCUR'05. These dramatic improvements come from a different internal architecture of the tool and improvements on the DBM library supporting subtractions, partitions, federations, intersections, specific operations for timed games (e.g. *predt*), and the essential operation of merging several difference bound matrices (DBMs) into one.

Fig. 1. Timed game automaton example.

In addition, the full input language of UPPAAL-4.0[BDH+06][2] is now supported, which gives the user access to C-like syntax to declare functions, custom
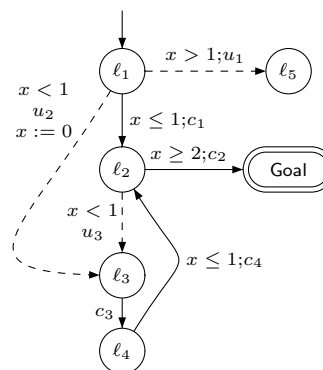
---

[1] http://www.cs.aau.dk/~adavid/tiga/
[2] See *http://www.uppaal.com*.

types, and other control statements (loops etc). The input language is given by a network of timed game automata [MPS95] (TGA).

Let us consider the example of the TGA A shown in Fig. 1 consisting of a timed automaton with one clock x and two types of edges: controllable (c_i) and uncontrollable (u_i). The reachability game consists in finding a strategy for a controller, *i.e.* when to take the controllable transitions that will guarantee that the system, regardless of when and if the opponent chooses to take uncontrollable transitions, will eventually end up in the location Goal.

```
State: ( A.l1 )
While you are in (A.x<1), wait.
When you are in (A.x==1), take transition A.l1->A.l2 { x <= 1, tau, 1 }

State: ( A.l2 )
While you are in (1<=A.x && A.x<2) || (A.x<1), wait.
When you are in (2<=A.x), take transition A.l2->A.Goal { x >= 2, tau, 1 }

State: ( A.l3 )
When you are in (A.x<=1), take transition A.l3->A.l4 { 1, tau, 1 }

State: ( A.l4 )
When you are in (A.x==1), take transition A.l4->A.l2 { x <= 1, tau, 1 }
While you are in (A.x<1), wait.
```

Fig. 2. Synthesized strategy of the example of Fig. 1.

Winning (or losing) conditions of the game are specified by TCTL formulas in the tool. In this example, we play a reachability game and the corresponding formula `control: A<> A.Goal` is satisfied. The tool gives the strategy (obtained with option `-t0`) shown in Fig. 2. The tool supports both reachability and safety games. If there is no winning strategy, the tool will give a counter strategy for the opponent (environment) to make the controller lose.

The new features of UPPAAL-TIGA since version 0.7 are:

– Better performance.
– Definition of winning/losing conditions as TCTL formulas.
– Support for both reachability and safety games.
– Support for the full input language of UPPAAL-4.0 (except for priorities that are incompatible).
– Output of strategies.
– Different compactness levels for the strategies (can output the strategy as a BDD/CDD).
– Ability to play against UPPAAL-TIGA with the command line verifier.

2 Strategy Synthesis

A *strategy* is a function $f : L \times \mathbb{R}^C_{\geq 0} \to \mathsf{Act}_c \cup \{\lambda\}$ that constantly gives information as to what the controller should do during the course of the game. In a given situation, the strategy could suggest the controller to either "do a particular controllable action" or "do nothing at this point in time (λ)". A strategy

is said to be a *winning strategy* if the controller supervised by the strategy always win the game whatever actions are chosen by the environment. If there is no such winning strategy, there exists a *counter-strategy* to either make the controller lose (reach a state marked as losing) or just prevent it to win.

2.1 Query Extensions

Winning conditions are specified through extensions of the UPPAAL query language. Given a timed game automaton $\mathcal{A}$, a set of goal states (`win`) and/or a set of bad states (`lose`), both defined by classical UPPAAL state formulas, four types of winning conditions can be issued. For all of them, the game is to find a controllable strategy f such that $\mathcal{A}$ supervised by f ensures that:

− control: A<> win (**must** reach `win`)
− control: A[not(lose) U win] (**must** reach `win` and **must** avoid `lose`)
− control: A[not(lose) W win] (**should** reach `win` and **must** avoid `lose`)
− control: A[] not(lose) (**must** avoid `lose`)

By default, UPPAAL-TIGA first check whether there is or not a winning strategy for the timed game given a winning condition. Note that it is always better to start asking for existence because the process of strategy extraction is quite demanding.

2.2 Strategy Extraction

The command line tool `verifytga` in the UPPAAL-TIGA package can output a strategy for each given winning condition when used with the `-t0` option. If no such strategy exists, a counter-strategy for the environment is given. Two additional options are available to control compactness: `-c0` makes sure that states with a given discrete part are unique and `-c1` prints the strategy as a BDD/CDD representation.

2.3 Game Simulation

The `-p` option of `verifytga` is the interactive mode where users can play against the strategy synthesized by UPPAAL-TIGA. Users may choose transitions to fire or to delay when it is their turns to play. For example, choices can be: `t2` (take transition t_2) or `w4.3` (wait for 4.3 time units).

3 Experiments And Conclusion

We compare the new verifier against the old prototype (old) against the current new version (new). The example is the production cell of [LL95,MW98]. Time constraints are defined to make the model controllable (c) or uncontrollable (u) with different numbers of plates (c3, c6 ...). Table 1 shows time consumption is given in seconds (s) and memory usage in megabytes (M). The old version

Model	c3		c6		c12		u3		u6		u12	
Old	0.1s	1M	12s	63M	-	-	0.2s	3M	235s	273M	-	-
New	0.05s	3.5M	0.05s	3.5M	0.14s	55M	0.02s	3.5M	0.04s	3.5M	0.12s	55M

Table 1. Experimental results on the production cell.

cannot handle the models with 12 plates, which is shown by "-". The new version is obviously far better.

The new version of Uppaal-Tiga is becoming mature and is fully operational with the GUI of Uppaal and the only missing feature of the tool is the ability to play strategies in the simulator, which we are currently working on. An interesting remark that we cannot develop in this short abstract is that back-propagating only winning or losing information is more efficient than back-propagating both as first thought in [CDF+05].

References

[BDH+06] Gerd Behrmann, Alexandre David, John Håkansson, Martijn Hendriks, Kim G. Larssen, Paul Pettersson, and Wang Yi. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems*, pages 125–126. IEEE Computer Society, 2006.

[CDF+05] F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient On-the-fly Algorithms for the Analysis of Timed Games. In *Proc. of 16th Int. Conf. on Concurrency Theory (CONCUR'05)*, volume 3653 of *LNCS*, pages 66–80. Springer, 2005.

[LL95] C. Lewerentz and T. Lindner. Production Cell: A Comparative Study in Formal Specification and Verification. In *Methods, Languages & Tools for Construction of Correct Software*, volume 1009 of *LNCS*, pages 388–416. Springer, 1995.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems. In *Proc. 12th Symp. on Theoretical Aspects of Computer Science (STACS'95)*, volume 900, pages 229–242. Springer, 1995.

[MW98] H. Melcher and K. Winkelmann. Controller Synthesis for the "Production Cell" Case Study. In *Proc. of 2nd Work. on Formal Methods in Software Practice*, pages 24–36. ACM Press, 1998.