IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems

Étienne André[1], Laurent Fribourg[2], Ulrich Kühne[3] and Romain Soulat[2]

[1]LIPN, CNRS UMR 7030, Université Paris 13, France
[2]LSV – ENS Cachan & CNRS
[3]Universität Bremen, Germany

Abstract. The tool IMITATOR implements the *Inverse Method* (*IM*) for Timed Automata (TAs). Given a TA $\mathcal{A}$ and a tuple π_0 of reference valuations for timings, *IM* synthesizes a constraint around π_0 where $\mathcal{A}$ behaves in the same discrete manner. This provides us with a quantitative measure of robustness of the behavior of $\mathcal{A}$ around π_0. The new version IMITATOR 2.5 integrates the new features of stopwatches (in addition to standard clocks) and updates (in addition to standard clock resets), as well as powerful algorithmic improvements for state space reduction. These new features make the tool well-suited to analyze the robustness of solutions in several classes of preemptive scheduling problems.

Keywords: Real-Time Systems, Parametric Timed Automata, Stopwatches

1 Motivation

IMITATOR 2.5 (for *Inverse Method for Inferring Time AbstracT behaviOR*) is a tool for parameter synthesis in the framework of real-time systems based on the inverse method *IM* for Parametric Timed Automata (PTAs). Different from CEGAR-based methods, this algorithm for parameter synthesis makes use of a "good" parameter valuation π_0 instead of a set of "bad" states [3]. IMITATOR takes as input a network of PTAs with stopwatches and a reference valuation π_0; it synthesizes a constraint K on the parameters such that (1) $\pi_0 \models K$ and (2) for all parameter valuation π satisfying K, the trace set (i.e., the discrete behavior) of $\mathcal{A}$ under π is the same as for $\mathcal{A}$ under π_0. This provides the system with a criterion of *robustness* (see, e.g., [14]) around π_0.

Fig. 1. Functional view of IMITATOR

History and New Features A basic implementation named IMITATOR has first been proposed, under the form of a Python script calling HYTECH [11]. The tool has then been entirely rewritten in IMITATOR II [2], under the form of a standalone OCaml program. A number of case studies containing up to 60 timing parameters could be efficiently verified in the purely timed framework.

Since [2], we extended the input formalism to PTAs equipped with *stopwatches*: clocks can now be stopped for some time while others keep growing. Also, we added clock updates: clocks can now be set to arbitrary linear combinations of other clocks, parameters and discrete variables. These extensions, together with powerful algorithmic improvements for state space reduction, allow us to consider larger classes of case studies, such as scheduling problems.

2 Architecture and Features

The core of IMITATOR (available under the GNU GPL license) is written in OCaml, and interacts with the Parma Polyhedra Library (PPL) [6]. Exact arithmetics with unbounded precision is used. IMITATOR takes as input a network of PTAs with stopwatches. The input syntax allows the use of clocks (or stopwatches), rational-valued discrete variables, and parameters (i.e., unknown constants) to be used altogether in linear terms, within guards, invariants and updates. A constraint is output in text format; furthermore, the set of traces computed by the analysis can be output under a graphical form (using Graphviz) for case studies with reasonable size (up to a few thousands reachable states).

IMITATOR implements in particular the following algorithms:

Full reachability analysis Given a PTA, it computes the reachability graph.

Inverse method Given a PTA and a reference parameter valuation π_0, it computes a constraint K on the parameter guaranteeing the same time-abstract behavior as under π_0 (see Figure 1).

IMITATOR 2.5 makes use of several algorithmic optimizations. In particular, we implemented a technique that merges any two states sharing the same discrete part and such that the union of their constraint on the clocks and parameters is convex [5]. This optimization preserves the correctness of all our algorithms; better, the output constraint is then always weaker or equal, i.e., covers a set of parameter valuations larger or equal. It behaves particularly well in the framework of scheduling problems, where the state space is drastically reduced. Actually, most of the scheduling examples we consider run out of memory without this merging technique.

3 Application to Robustness Analysis in Scheduling

Due to the aforementioned state space reduction and the use of stopwatches, IMITATOR 2.5 becomes an interesting tool for synthesizing robust conditions for scheduling problems. Let us illustrate this on a preemptive jobshop example

given in [1]. The jobshop scheduling problem is a generic resource allocation problem in which common resources ("machines") are required at various time points (and for given duration) by different tasks. For instance, one needs to use a machine m_1 for d_1 time units, machine m_2 for d_2 time units, and so on. The goal is to find a way ("schedule") to allocate the resources such that all tasks terminate as early as possible ("minimal makespan"). Let us consider the jobshop problem $\{J_1, J_2\}$ for 2 jobs and 3 machines with: $J_1 = (m_1, d_1), (m_2, d_2), (m_3, d_3)$ and $J_2 = (m_2, d'_2)$ with $d_1 = 3, d_2 = 2, d_3 = 4, d'_2 = 5$. There are many possible schedules (two of them are depicted in Figure 2 in Appendix). In [1], this problem is modeled as a product $\mathcal{A}$ of TAs with stopwatches, each TA modeling a job. Each schedule corresponds to a branch in the reachability tree of $\mathcal{A}$. The makespan value corresponds to the duration of the shortest branch, here 9.

Let us explain how to analyze the robustness of the valuation $\pi_0 : \{d_2 = 2, d'_2 = 5\}$ with respect to the makespan value 9. We first consider a parametric version of $\mathcal{A}$ where d_2 and d'_2 become parameters. In the same spirit as in [9], we add an observer $\mathcal{O}$, which is a TA synchronized with $\mathcal{A}$, that fires a transition labeled *DEADLINE* as soon as a schedule spends more than 9 time units. We then use IMITATOR (instead of a CEGAR-like method as in [9]) with $\mathcal{A} \parallel \mathcal{O}$ as a model input and π_0 as a valuation input. This yields the constraint K: $7 > d'_2 \wedge 3 > d_2 \wedge d'_2 + d_2 \geq 7$. (For a geometrical representation of K, see Figure 3 of Appendix 1.) By the *IM* principle, the set of traces (i.e., discrete runs) of $\mathcal{A} \parallel \mathcal{O}$ is always the same, for any point (d_2, d'_2) of K. (This set of traces is depicted under the form of a tree in Figure 4 in Appendix 1.) Since the makespan for π_0 is 9, we know that some branches of the tree do not contain any *DEADLINE* label (these branches end at node $s73$ in Figure 4). This holds for each point (d_2, d'_2) of K. The makespan of the system is thus always at most 9 in K. (In particular, we can increase d_2 from 2 to 3, or increase d'_2 from 5 to 7 while keeping the makespan less than or equal to 9.)

Another example of application of IMITATOR 2.5 to a scheduling problem, originating from [8], is given in Appendix 2 (using the cartography method of [4]). All case studies and experiments are described in a research report [15]. See also `http://www.lsv.ens-cachan.fr/Software/imitator/`.

4 Comparison with Related Work

The use of models such as PTAs and parametric Time Petri Nets (TPNs) for solving scheduling problems has received attention in the past few years. For example, Roméo [13] performs model checking for parametric TPNs with stopwatches, and synthesizes parameter valuations satisfying TCTL formulæ. An extension of UPPAAL allows parametric model checking [7], although the model itself remains non-parametric. The approach most related to IMITATOR 2.5 is [9, 12], where the authors infer parametric constraints guaranteeing the feasibility of a schedule, using PTAs with stopwatches. The main difference between [9, 12] and IMITATOR relies in our choice of the inverse method, rather than a CEGAR-based method. First results obtained on the same case studies are incomparable (although sim-

ilar in form), which seems to indicate that the two methods are complementary. The problem of finding the schedulability region was attacked in analytic terms in [8]; the size of our examples is rather modest compared to those treated using such analytic methods. However, in many schedulability problems, no analytic solution exists (see, e.g., [16]), and exhaustive simulation is exponential in the number of jobs. In such cases, symbolic methods as ours and those of [9, 12] are useful to treat critical real-life examples of small size. We are thus involved in a project [10] with an industrial partner with first interesting results.

References

1. Y. Abdeddaïm and O. Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, pages 113–126, 2002.
2. É. André. IMITATOR II: A tool for solving the good parameters problem in timed automata. In *INFINITY*, volume 39 of *EPTCS*, pages 91–99, 2010.
3. É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.
4. É. André and L. Fribourg. Behavioral cartography of timed automata. In *RP*, volume 6227 of *LNCS*, pages 76–90. Springer, 2010.
5. É. André, L. Fribourg, and R. Soulat. Enhancing the inverse method with state merging. In *NFM*, volume 7226 of *LNCS*. Springer, 2012. To appear.
6. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.
7. G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Beyond liveness: Efficient parameter synthesis for time bounded liveness. In *FORMATS*, pages 81–94, 2005.
8. E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Computers*, 53(11):1462–1473, 2004.
9. A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS*, pages 80–89, Washington, DC, USA, 2008. IEEE Computer Society.
10. L. Fribourg and D. Lesens. Projet ROSCOV: Robuste ordonnancement de systèmes de contrôle de vol. Project report (in French), December 2011. Available at `www.farman.ens-cachan.fr/ROSCOV.pdf`.
11. T. A. Henzinger, P. H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.
12. T. Le, L. Palopoli, R. Passerone, Y. Ramadian, and A. Cimatti. Parametric analysis of distributed firm real-time systems: A case study. In *ETFA*, pages 1–8, 2010.
13. D. Lime, O. H. Roux, C. Seidner, and L.-M. Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In *TACAS*, volume 5505 of *LNCS*, pages 54–57. Springer, 2009.
14. N. Markey. Robustness in real-time systems. In *SIES*, pages 28–34. IEEE, 2011.
15. R. Soulat. Scheduling with IMITATOR: Some case studies. Research Report LSV-12-05, Laboratoire Spécification et Vérification, France, March 2012. Available on `www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2012-05.pdf`.
16. J. Sun, M. K. Gardner, and J. W. S. Liu. Bounding completion times of jobs with arbitrary release times, variable execution times, and resource sharing. *IEEE Trans. Softw. Eng.*, 23:603–615, 1997.

Appendix 1: Jobshop Case Study from [1]

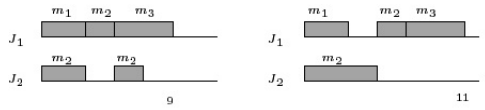

Fig. 2. Schedules for jobshop example (with makespan 9 (left) and 11 (right)) [1]

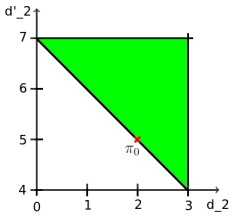

Fig. 3. Geometrical representation of K around $\pi_0 : (2, 5)$

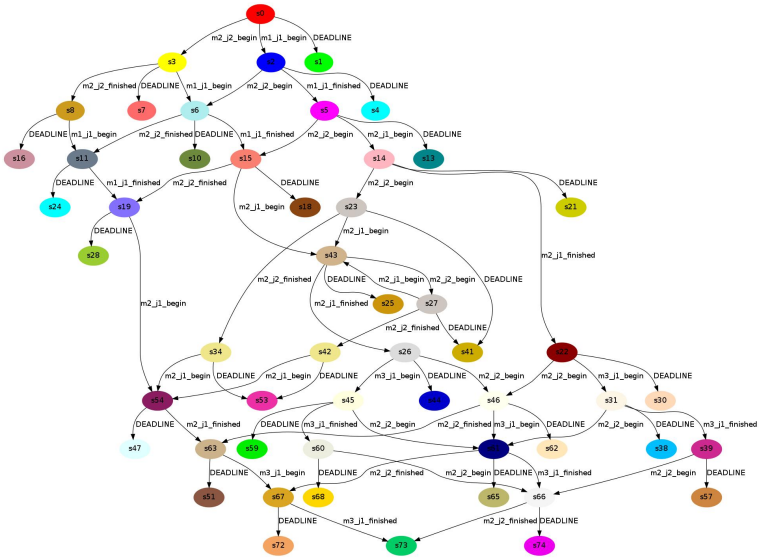

Fig. 4. Graphics of the trace set output by IMITATOR on the jobshop example

Appendix 2: Rate Monotonic Case Study from [8]

We present here what we can obtain from IMITATOR 2.5 with the Behavioral Cartography (BC) output. By iterating IM over all the integers points inside a finite but dense rectangle V_0 in the parametric space, one is able to decompose (most of) the parametric space included into V_0 into behavioral tiles[1] [4].

For a given tile, we ensure that for every valuation of the parameters in this tile, the behavior of $\mathcal{A}$ is the same; therefore, we only have to test the schedulability of a single point for each tile to ensure the schedulability on the whole tile.

Let us apply this method on a "Rate monotonic" example of [8, Section III]. There are three periodic tasks τ_1, τ_2 and τ_3 with periods of $T_1 = 3,\ T_2 = 8$ and $T_3 = 20$ and deadlines of $D_1 = 3,\ D_2 = 8$ and $D_3 = 20$. We are interested in finding the set of computation times of each task such that the system is schedulable. (The interested reader can find the full details in [8].)

We set V_0 as $C_1 \in [0, 3]$, $C_2 \in [0, 8]$ and $C_3 = 0$, where C_i is the computational time of τ_i. The system is unschedulable if there exists a task τ_i such that $C_i > T_i$. Algorithm BC outputs the set of tiles and we check for one point of each tile whether the system is schedulable or not. The result for this example is given in Figure 5 for this V_0 with a discretization step of 0.2.

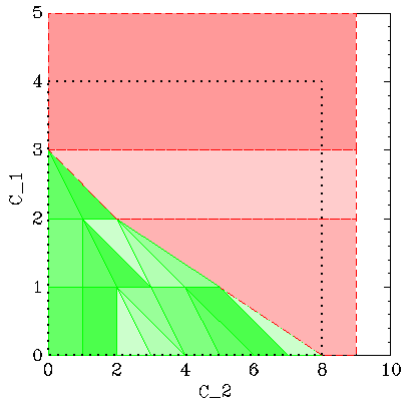

Fig. 5. Cartography output by IMITATOR 2.5 (in green the system is schedulable; in red, unschedulable)

Note that the full dense parametric space (including beyond V_0) is covered after only 20 applications of IM. The green part corresponds to the bottom left "triangle" of [8, Figure 1(a)].

[1] Actually, one can take a finer discretization step than integers to ensure a better coverability of V_0