# IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling

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 $\pi_0$ of reference valuations for timings, *IM* synthesizes a constraint around $\pi_0$ 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 $\pi_0$. 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, Scheduling, Parametric Timed Automata with Stopwatches, Parameter Synthesis

⇐ **Version avec commentaires** ⇒

## 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 (see [8]), this original semi-algorithm is based on a "good" parameter valuation $\pi_0$ instead of a set of "bad" states [4]. IMITATOR takes as input a network of TAs with stopwatches; it synthesizes a constraint $K$ on the parameters such that (1) $\pi_0 \models K$ and (2) for all parameter valuation $\pi$ satisfying $K$, the trace set (i.e., the discrete behavior) of $\mathcal{A}$ under $\pi$ is the same as for $\mathcal{A}$ under $\pi_0$. This preserves in particular linear time properties, and provides the system with a criterion of *robustness* (see, e.g., [11]), by formally guaranteeing the system correctness around the reference parameter valuation $\pi_0$.

**History and New Features** A basic implementation named IMITATOR has first been proposed, under the form of a Python script calling HYTECH. The tool has then been entirely rewritten in IMITATOR II [3], 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.


**Fig. 1.** Functional view of IMITATOR

Since [3], we extended the input formalism to parametric timed automata 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. We also implemented further algorithms based on *IM*, that satisfy weaker properties than the preservation of the trace sets, while outputting larger sets of parameter valuations than *IM* (and possibly non-convex). These extensions allow us to consider larger classes of case studies, and in particular scheduling problems.

## 2 Architecture and Features

IMITATOR is available under the GNU GPL license. The core of the program 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 TAs with stopwatches, that synchronize on shared actions. The input syntax, inspired by HYTECH, 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 reasonably sized case studies (up to a few thousands reachable states).

IMITATOR implements in particular the following algorithms:

**Full reachability analysis** Given a model, it computes the reachability graph.
**Inverse method** Given a model and a reference parameter valuation $\pi_0$, it computes a constraint on the parameter guaranteeing the same time-abstract behavior as under $\pi_0$.

IMITATOR makes use of several algorithmic optimizations. In particular, we implemented a technique that merge 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 constraint output by *IM* in that case is always weaker or equal, i.e., covers a set of parameter valuations larger or equal. This optimization 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 optimization.

## 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 [2]. 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 with 2 jobs on 3 machines: $\{J_1, J_2\}$ 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 illustrated in Appendix 4. In [2], this problem is modeled as a product $\mathcal{A}$ of TAs with stopwatch, each TA corresponding to 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, and is equal here to 9.

Suppose that we want to analyse 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 [7], we add an observer $\mathcal{O}$, which is a TA synchronized with $\mathcal{A}$, which fires a transition labeled *DEADLINE* as soon as a schedule spends more than 9 units of time. We then use IMITATOR (instead of a CEGAR-like method as in [7]) with $\mathcal{A} \parallel \mathcal{O}$ as a model input and $\pi_0$ 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 Appendix 4). 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 Appendix 4. Since the makespan for $\pi_0$ is 9, we know that there is at least one branch of the tree without any *DEADLINE* label. This holds for all the points $(d_2, d'_2)$ of $K$. In other words, the makespan of the system is (at most) equal to 9, for every point of $K$.

Such a robustness analysis, inspired from [7], has been conducted with IMITATOR 2.5 on several classes of scheduling problems (see [12] for details).

## 4 Related Work

The use of PTAs and their extensions for scheduling problems has received attention in the past few years (see, e.g., Section 7 of [7] for a survey). The approach most related to IMITATOR 2.5 is [7, 10], where the authors infer parametric constraints guaranteeing the feasibility of a schedule, using PTAs with stopwatches. The main difference between [7, 10] and IMITATOR relies in our choice of the inverse method, rather than a CEGAR-based method. Also, although IMITATOR offers facilities for robust analysis of scheduling problems, the tool is not specifically dedicated to scheduling, but applies to the verification of many classes of

concurrent real-time systems, such as communication protocols and hardware components (see, e.g., [4]).

**Future Work** In [9], the inverse method has been extended to hybrid automata, and a prototype based on IMITATOR has been implemented. Bridging the gap between that experimental prototype with high expressiveness but moderate performances, and the optimized IMITATOR 2.5 dedicated to PTAs with stopwatches is the subject of ongoing work.

## References


1. IMITATOR's Web page. `http://www.lsv.ens-cachan.fr/Software/imitator/`.
2. Y. Abdeddaïm and O. Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, pages 113–126, 2002.
3. É. André. IMITATOR II: A tool for solving the good parameters problem in timed automata. In *INFINITY'10*, volume 39 of *EPTCS*, pages 91–99, 2010.
4. 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.
5. É. André, L. Fribourg, and R. Soulat. Enhancing the inverse method with state merging. In A. Goodloe and S. Person, editors, *NFM'12*, volume 7226 of *LNCS*, pages 100–105. 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. A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS'08*, pages 80–89, Washington, DC, USA, 2008. IEEE Computer Society.
8. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV'00*, pages 154–169. Springer-Verlag, 2000.
9. L. Fribourg and U. Kühne. Parametric verification and test coverage for hybrid automata using the inverse method. In G. Delzanno and I. Potapov, editors, *RP'11*, volume 6945 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2011.
10. T. T. H. 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.
11. N. Markey. Robustness in real-time systems. In *SIES*, pages 28–34. IEEE, 2011.
12. R. Soulat. Scheduling with IMITATOR: Some case studies. 2012.
13. 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

## Visual Examples of Schedules

Figure 2. ⇐ **?** ⇒

⇐ **?** ⇒

**Fig. 2.** Schedules

## Geometrical Representation of $K$

⇐ **?** ⇒

**Fig. 3.** Geometrical Representation of $K$

## Summary of Experiments

**⇐ ajouter un beau tableau avec des contraintes, et un lien vers un rapport d'etudes de cas (+ site Web) ⇒**

| Case study | $\|\mathcal{A}\|$ | $\|X\|$ | $\|P\|$ | $\|S\|$ | $\|T\|$ | $n$ | $\|K\|$ | t |
|---|---|---|---|---|---|---|---|---|
| Astrium (FP)[12] | 5 | 7 | 10 | 63/63 | 62/62 | 32/32 | 12/12 | 1.10/0.93 |
| FP 1 | 5 | 7 | 10 | 205/805 | 225/837 | 40/40 | 11/11 | 15.6/28.5 |
| FP 2 | 5 | 7 | 10 | 208/1174 | 228/1269 | 42/42 | 12/12 | 10.7/41.0 |
| FP 3 | 5 | 7 | 10 | 49/82 | 51/81 | 15/15 | 12/12 | 1.03/1.75 |
| Astrium (EDF) | 5 | 10 | 13 | 63/63 | 62/62 | 32/32 | 13/13 | 2.42/2.30 |
| EDF 1 | 5 | 10 | 13 | 76/415 | 91/454 | 31/31 | 20/20 | 66.1/288.2 |
| EDF 2 | 5 | 10 | 13 | 254/642 | 326/817 | 47/47 | 12/12 | 9.9/23.2 |
| EDF 3 | 5 | 10 | 13 | 31/43 | 33/45 | 9/9 | 13/14 | 1.09/1.57 |
| cpr08[7] | 4 | 6 | 8 | 676/3194 | 886/3931 | 15/15 | 15/10 | 288.2/450.4 |
| HPPR10[10] | 3 | 4 | 9 | 60/103 | 103/686 | 10/10 | 7/5 | 2.10/11.7 |
| Task chains[13] | 8 | 15 | 18 | 215/1364 | 264/1363 | 15/15 | 17/17 | 85.3/270.6 |

**Fig. 4.** Title

**⇐ quand il y a une variablité du computation time (entre BCET et WCET), pas de méthode exacte analytique –¿ interet des methodes symboliques (model checking) d'exploration exhaustive basee sur TA**

**(et donc interet de IM puisque generalisation! si on etait schedulable, on lest encore avec IM) ⇒**

Astrium, FPs and EDFs case studies are composed of three cyclic tasks that need to be performed on a single machine. Therefore, the machine has to choose which task is active when several tasks are requiring computation. This is done in two ways, either with a Fixed Priority (FP) scheduler or an Earliest Deadline First scheduler (EDF). These three case studies are being studied in the framework of the joint project Roscov between LSV and Astrium.

cpr08 [7] and HPPR10 [10] are two-cyclic-tasks problem with an EDF scheduler. TaskChain [13] does not use cyclic tasks but instead has a 4 task chain and a concurrent 3 task chain. Each task in a chain cannot start until the previous one has not terminated. Each task in each chain has a given fixed priority. A scheduler assigns which task has the computational time with a FP policy.

For every FP scheduling example, the parameters are the offsets of the tasks, their Wost Case Execution Time (WCET), their periods and a deadline to achieve a given goal (modeled by an extra automaton). For every EDF scheduling example, the parameters are the offsets of the tasks, their WCET, their periods, a deadline to achieve each task and a deadline to achieve a given goal (modeled by an extra automaton).

Experiments were performed on Ubuntu 11.10 equipped with an Intel Core 2 2.93 GiHz processor with 2 GiB RAM. Source code, binaries, all case studies and logs are available in [1].

### Example of Trace Set (Jobshop example of [2] with 2 jobs and 3 machines)

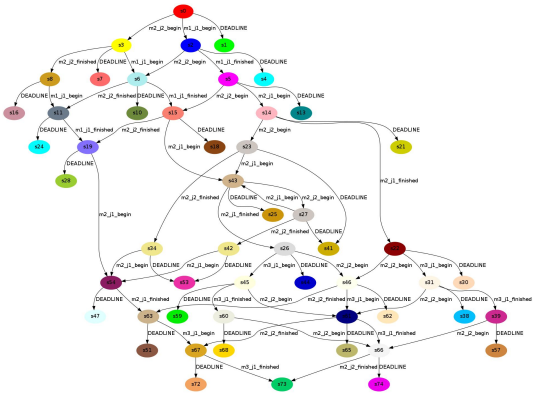

**Fig. 5.** Examples of graphics output by IMITATOR

**⇐ commenter figure: maler, merging; seul l'etat du bas respecte ⇒**
**⇐ un automate par tache, un scheduler ⇒**

⇐ ex de TA en appendix, avec schema et code IMITATOR ⇒

### Example of Trace Set

⇐ ajouter un trace set en mode fancy ⇒

### An Example of Model

⇐ ajouter un modele en entree avec sa representation graphique (pour decrire la syntaxe) ⇒