IMITATOR 3: Synthesis of Timing Parameters Beyond Decidability

Étienne André(✉)

Université de Lorraine, CNRS, Inria, LORIA, 54000 Nancy, France
Andre.Etienne@lipn13.fr

Abstract. Real-time systems are notoriously hard to verify due to nondeterminism, concurrency and timing constraints. When timing constants are uncertain (in early the design phase, or due to slight variations of the timing bounds), timed model checking techniques may not be satisfactory. In contrast, parametric timed model checking synthesizes timing values ensuring correctness. IMITATOR takes as input an extension of parametric timed automata (PTAs), a powerful formalism to formally verify critical real-time systems. IMITATOR extends PTAs with multi-rate clocks, global rational-valued variables and a set of additional useful features. We describe here the new features and algorithms offered by IMITATOR 3, that moved along the years from a simple prototype dedicated to robustness analysis to a standalone parametric model checker for timed systems.

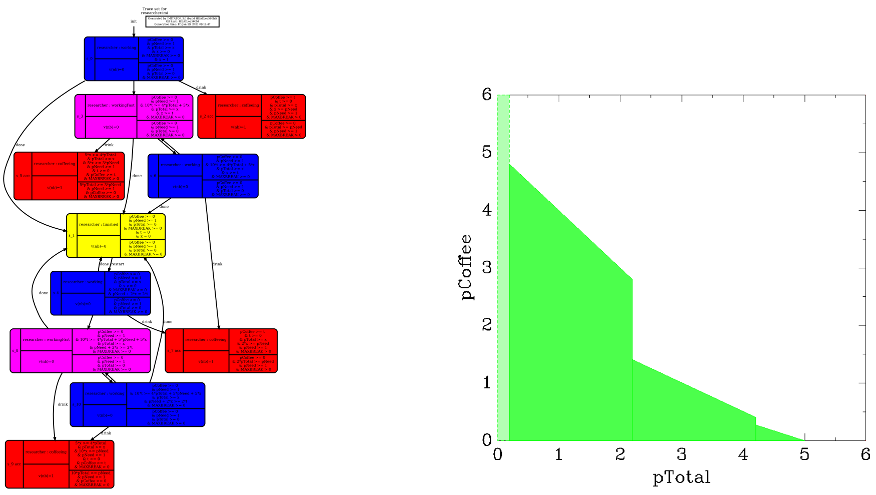
Keywords: Parametric timed automata · Parameter synthesis · Real-time systems

1 Introduction

Real-time systems are often used in critical environments, and may be verified using formal methods. Such systems are notoriously hard to verify due to nondeterminism, concurrency and timing constraints. Timed model checking provides designers with techniques to formally verify a real-time system. However, timed model checking may not always be fully satisfactory: First, in the early design phase, timing constants may not be known and, without them, model checking is not possible; Second, at runtime, timing constants may vary (due to uncertain bounds, or to processor clock drifts), in which case the model checking result may not hold anymore. In contrast, parametric timed model checking *synthesizes* timing values ensuring the system correctness.

Parametric timed automata (PTAs) are a powerful formalism to reason about, and formally verify critical real-time systems [5]. PTAs are finite state

This work is partially supported by the ANR-NRF French-Singaporean research program ProMiS (ANR-19-CE25-0015).



(a) State space (b) 2-dimensional constraint

Fig. 1. Examples of graphical outputs

automata extended with clocks, i.e., real-valued variables evolving linearly, that can be compared with either integer constants or parameters in guards (constraints to take a transition) and invariants (constraints to remain in a location).

IMITATOR takes as input networks of "IMITATOR PTAs" (IPTAs) extending PTAs with several convenient features such as stopwatches, multi-rate clocks or global shared rational-valued variables.

IMITATOR answers variants of the following problem:

Parameter synthesis problem:
INPUT: A network of IPTAs $\mathcal{A}$ and a specification φ
PROBLEM: Synthesize the set of parameter valuations for which $\mathcal{A}$ satisfies φ

IMITATOR answers this problem by synthesizing sets of parameter valuations in the form of a finite disjunction of linear constraints over the parameters.

IMITATOR is a command-line only tool; its input is text-based (partially inspired by HyTech syntax [41]) and is "human-readable", different from, e.g., XML. IMITATOR produces standardized result files (that can be possibly parsed from external tools), and can produce graphical outputs, such as in Fig. 1.

The expressive power (i.e., ease to write a complicated model in a compact manner) of the tool has been largely improved since IMITATOR 2.5 [17], and IMITATOR is now a parametric timed model checker taking as inputs a model and a property, implementing various synthesis algorithms.

2 An Expressive Input Language

Parametric Timed Automata (PTAs). Timed automata (TAs) [3] extend finite-state automata with *clocks*, i.e., real-valued variables evolving at the same rate 1, that can be compared to integers along edges ("guards") or within locations ("invariants"). Clocks can be reset (to 0) along transitions. PTAs extend TAs with (timing) parameters, i.e., unknown rational-valued constants [5].

Example 1. In the model in Fig. 2 (that goes beyond the syntax of PTAs, see Example 2), there are four locations, depicted as rounded rectangles. Invariants are depicted using dotted rectangles. In the invariant of location working, clock x is compared to parameter p_{total}. The guard of the transition from coffee to working compares clock t to p_{coffee}; this clock t is reset to 0 along this transition.

IMITATOR *Parametric Timed Automata (IPTAs).* IMITATOR takes as input models described as networks of IMITATOR parametric timed automata (IPTAs). IPTAs extend PTAs with a set of useful features, described in the following.

Global Rational-Valued Variables. Global variables (called "discrete") can be defined, and are part of the discrete part of a state, together with locations (and different from clocks and parameters that are part of the *continuous* part). Global variables in IMITATOR are exact rationals, following exact arithmetics (as opposed to, e.g., floating-point arithmetic that can accumulate errors and lead to faulty assertions). Exact rationals are encoded in IMITATOR using the GNU MP library. Such discrete variables can be updated along transitions, and can also be part of the clock guards and invariants; in fact, virtually any linear expression over clocks, parameters and discrete variables can be used in guards, invariants and updates. Non-linear arithmetic expressions over sole discrete variables are allowed too.

Automata Synchronization. IPTAs can be synchronized together on shared actions, or by reading shared variables. All variables (clocks, parameters, discrete) are potentially global in IMITATOR. This allows users to define models component by component.

Arbitrary Flows. Since version 3.0, IMITATOR supports arbitrary (constant) flows for clocks; this way, clocks do not necessarily evolve at the same time, and can encode different concepts from only time: temperature, amount of completion, continuous cost... Their value can increase or decrease at any predefined rate in each location, and can become negative. In that sense, IMITATOR's clocks are closer to *continuous variables* (as in hybrid automata) rather than TAs' clocks; nevertheless, we keep the name *clock* for sake of backward-compatibility. This makes IMITATOR support a parametric extension of *multi-rate automata* [2]. This notably includes stopwatches, where clocks can have a 1- or 0-rate [36].

Additional Syntax Improvements. Beyond the aforementioned increase of the syntactic expressive power, the syntax was enhanced with accepting locations (that can be used in properties), global constants, "`if... then... else`" conditions in updates, and with the ability to include model fragments from different

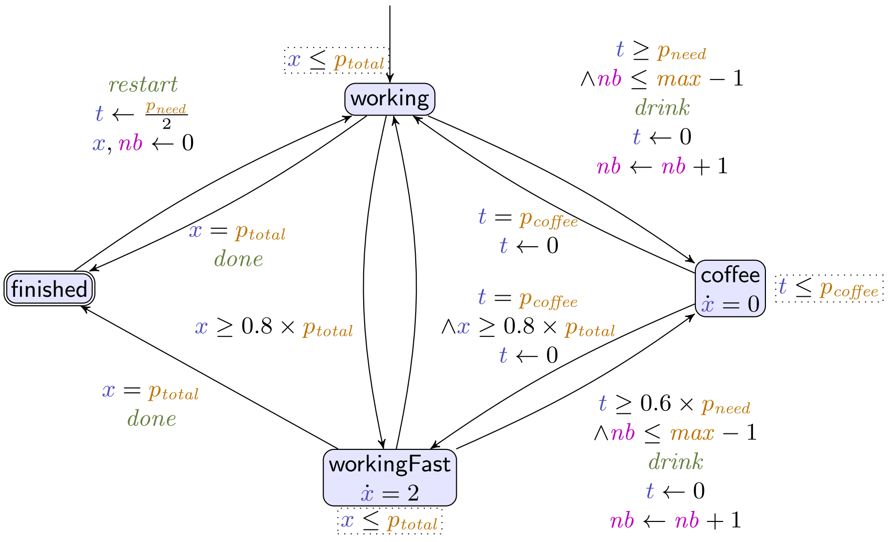


Fig. 2. An IPTA example: writing papers and drinking coffee

files (new syntax `#include(modelpart.imi)`). Several simplifications were made to the syntax to keep it "human-readable". For example, location workingFast of Fig. 2 is written in IMITATOR syntax as follows:

```
loc workingFast: invariant x <= pTotal flow{x' = 2}
```

Translations. Finally, translations of the model are available to other model checkers such as HyTech [41] and Uppaal [42] (in both cases, not all features can be translated since some of the features of IMITATOR do not exist in the target tool, e.g., Uppaal does not support parameters nor complex linear constraints over clocks (only "diagonal")). Graphical translations of the model are also available to JPEG, PDF and LaTeX formats.

Example 2. Consider the IPTA in Fig. 2, modeling a researcher writing papers. The model features two clocks t (measuring the time when needing a coffee) and x (measuring the amount of work done on a given paper), both initially 0. Their rate is always 1, unless otherwise specified (e.g., in workingFast). Initially, the researcher is working (location working) on a paper, requiring an amount of work p_{total}. When the paper is completed (guard $x = p_{total}$), the IPTA moves to location finished. From there, at any time, the researcher can start working on a new paper (transition back to working, updating x and t).

Alternatively, after at least a certain time (guard $t \geq p_{need}$), the researcher may need a coffee; this action can only be taken until a maximum number of coffees have been drunk for this paper ($nb \leq max - 1$), where nb is a discrete global variable recording the number of coffees drunk while working on the current paper. When drinking a coffee (location coffee), the work is obviously not progressing ($\dot{x} = 0$). Drinking a coffee takes exactly p_{coffee} time units (guard $t = p_{coffee}$ back to location working). Observe that, from the second paper onwards (transition labeled with *restart*), the researcher is already half-way of her/his need for a coffee (parametric update $t \leftarrow 0.5 \times p_{need}$ [22]).

Also, whenever 80% of the work is done (guard $x \geq 0.8 \times p_{total}$), the researcher may work twice as fast (location workingFast, with a rate 2 for clock x). In that case, (s)he needs a coffee faster too ($0.6 \times p_{need}$).

All three durations p_{coffee}, p_{need} and p_{total} are timing parameters. We fix their parameter domains as follows: $p_{coffee}, p_{total} \in [0, \infty)$ and $p_{need} \in [1, \infty)$. The maximum number of coffees $max \in [0, \infty)$ is also a parameter; observe that it is (only) compared to the discrete variable nb, and therefore can be seen as a "discrete parameter"—which is allowed by the liberal syntax of IMITATOR.

The example in Fig. 2 could not be modeled with UPPAAL due to the presence of timing parameters, stopwatches, multi-rate clocks and non-0 update. It may be modeled using HYTECH; however, most algorithms implemented in IMITATOR (even the most basic ones, such as liveness synthesis) do not exist in HYTECH, as HYTECH mainly focuses on basic state space computation.

3 A Variety of Synthesis Algorithms

The formalism of networks of IPTAs is "highly undecidable" for most problems. Indeed, while several problems are decidable for timed automata (notably the reachability [3]), most interesting problems become undecidable in the presence of timing parameters [5,8] , notably when such parameters are unbounded [35]. On top of this, multi-rate automata together with linear constraints over the clocks also yield undecidability [2]. Finally, the mere use of stopwatches, even without the aforementioned extensions, brings undecidability [36]. Also note that, in contrast to several existing model checkers, IMITATOR offers the use of unbounded rational variables, therefore with an infinite domain. For all these reasons, it is always possible to find examples of IPTAs for which the algorithms implemented in IMITATOR would not terminate with an exact (sound and complete) result. The rational behind IMITATOR is therefore to follow a "best-effort" approach, by:

– using aggressive optimizations and abstractions (e.g., [11,19,45]), leading to termination for most case studies in practice;
– outputting over- or under-approximated results, i.e., the set of synthesized parameter valuations may be larger or smaller than the exact result.

IMITATOR outputs a standardized result (in a text file), that contains the synthesized constraint with a set of information, and notably the *validity* of the constraint, i.e., whether the set of valuations is *exact* (sound and complete), *possibly over-approximated*, *possibly under-approximated*, or *potentially invalid* i.e., when both under-approximating and over-approximating heuristics were used.

By default, IMITATOR attempts to synthesize an exact result; only when some specific options are used (e.g., a limit on the number of states explored, or on the computation time), approximations may be used. These approximations are conservative for most algorithms; for example, if an approximation is used for safety synthesis, then the result will be under-approximated (i.e., the system is safe for all synthesized valuations—even though some more safe valuations may exist).

IMITATOR offers two main classes of synthesis: *i) Witness* (or counter-example), which attempts to exhibit at least one parameter valuation satisfying the property; often, IMITATOR still outputs a *symbolic* set of valuations (i.e., a linear constraint over the parameters), but stops the analysis as soon as one such set is found. *ii)* Normal *synthesis*, where IMITATOR attempts to synthesize *all* parameter valuations satisfying the property.

Properties include reachability (denoted by "`EF`", following the TCTL syntax), safety (denoted by "`AGnot`"), liveness, deadlock-freeness, robustness, and some others.

Throughout this section, we exemplify the main synthesis algorithms of IMITATOR on Example 2.[1] All the results synthesized in the following are exact (sound and complete), unless otherwise specified.

Safety. A first algorithm of IMITATOR is safety synthesis, i.e., synthesizing parameter valuations for which a discrete state (location and/or valuation of the discrete variables) is unreachable for all runs. For example, one synthesize the valuations for which it is impossible to drink any coffee, i.e., it is impossible to reach the coffee location of the "researcher" automaton of Fig. 2.

```
#synth AGnot(loc[researcher] = coffee)
```

$$\text{The result is: } max \in [0,1) \vee \left(max \geq 1 \wedge p_{total} < \frac{p_{need}}{10}\right)$$

Let us explain this result. The first disjunct is trivial: if the researcher is not allowed to drink any coffee ($max < 1$), the transition to coffee (guarded by "$nb \leq max - 1$") can never be taken. The second disjunct is, despite the relative simplicity of this model, less trivial: assume for illustration that $p_{need} = 10$ and $p_{total} = 1$, and let us show that the researcher is still able to start drinking a coffee in this situation. After the first paper completion (action $restart$), we have $x \leftarrow 0$ and $t \leftarrow 5$. After one time unit in location working ($x = 1$ and $t = 6$), the researcher moves to workingFast, and can immediately move to coffee (guard $t \geq 0.6 \times p_{need}$ is now satisfied). This scenario, that can be seen on the parametric state space output by IMITATOR (see Fig. 1a), is also possible for larger values of p_{total}. This explains the strict inequality $p_{total} < \frac{p_{need}}{10}$.

[1] All finishing executions for our example using IMITATOR 3.0 "Cheese" ea560fd on a Dell XPS 13 7390 Intel® Core^{TM} i7-10510U CPU @ 1.80 GHz running Linux Mint 20 Ulyana terminate within $< 1\,s$. All examples and results can be found at [9].

Reachability. Reachability can be seen as the opposite of safety, i.e., the goal is to synthesize parameter valuations for which a discrete state is reachable for at least one run. For example, one can ask for the valuations for which it is possible to drink at least one coffee:

```
#synth EF(loc[researcher] = coffee)
```

The result is $max \geq 1 \wedge p_{total} \geq \frac{p_{need}}{10}$, which is obviously the complement of the result synthesized for the aforementioned safety property.

One can also synthesize valuations for which it is possible to drink at least five coffees while working on some article (i.e., $nb \geq 5$).

```
#synth EF(loc[researcher] = coffee & nb >= 5)
```

The result is $max \geq 5 \wedge p_{total} \geq \frac{37}{10} \times p_{need}$.

Minimum-Time Reachability. Minimal-time synthesis [12] aims at synthesizing parameter valuations minimizing the time needed to reach a discrete state. Here, we can ask for the valuations for which it is possible to finish an article after drinking at least 2 coffees:

```
#synth EFtmin(loc[researcher] = finished & nb >= 2)
```

The result is $\frac{p_{total}}{2} + p_{need} + 2 \times p_{coffee} \leq 2 \wedge max \geq 2$ and the minimal time is 2. That is, any of these valuations guarantee the reachability of a state where the researcher has drunk 2 coffees, and the minimum time is 2 (recall that $p_{need} \in [1, \infty)$).

Optimal Parameter Reachability. One can ask here for the valuations for which the value of a given parameter is minimized or maximized when reaching a given state. Let us ask for the valuations minimizing the value of p_{total} when finishing a paper after drinking (at least) 3 coffees.

```
#synth EFpmin(loc[researcher] = finished & nb >= 3, pTotal)
```

The result is $max \geq 3 \wedge p_{total} = 2.1 \wedge p_{need} = 1$. Observe that p_{coffee} is not involved in this constraint (contrarily to minimum-time synthesis); indeed, the time spent in drinking coffee does not impact the total duration of the *work* (p_{total}), as the progress of clock x is stopped in coffee.

Parametric Deadlock Freeness. Deadlocks are states in which no discrete action can be taken, and time cannot elapse ("timelock"). Such situations may denote ill-formed models. IMITATOR offers an algorithm [7] synthesizing parameter valuations for which the model is deadlock-free. In case of "early termination" (predefined bound on the depth of the state space or on the computation time), a backward procedure synthesizes a subset of correct (deadlock-free) valuations.

```
#synth DeadlockFree
```

For this property, the analysis does not terminate, as the state space is infinite (unbounded rational-valued parameters, unbounded variable nb) and IMITATOR needs to explore it as a whole to deduce deadlock-freeness for our example.

Adding a bound on the depth of the state space (option `-depth-limit 40`) yields termination, with a *pair* of constraints: an under-approximated positive constraint (i.e., valuations that are guaranteed to be deadlock-free) $max < 16 \lor (max \geq 16 \land p_{total} < \frac{27}{2} p_{need})$, and an over-approximated negative constraint (i.e., valuations that *might* be deadlocked) $max \geq 16 \land p_{total} \geq \frac{27}{2} p_{need}$. Observe that both constraints are complementary, i.e., IMITATOR is sure that the former set is deadlock-free, and is not sure that the latter set contains deadlocks. (Note that, in fact, the model is very likely to be deadlock-free for all valuations, even though IMITATOR is not able to show it.)

Liveness Synthesis. A new feature of IMITATOR 3 is cycle synthesis, i.e., parameter valuations for which there exists an infinite run, possibly passing infinitely often by a given discrete state (Büchi condition). IMITATOR uses by default an original algorithm by Laure Petrucci and Jaco van de Pol based on NDFS extended with parametric subsumption and pruning [45] (other algorithms, such as BFS, are also available [11]). In our running example , one can ask for the valuations for which the researcher infinitely often writes papers after drinking (at least) 3 coffees for each of them.

```
#synth CycleThrough(loc[researcher] = finished & nb>=3)
```

The result is $max \geq 3 \land p_{total} \geq 2.1 \times p_{need}$.

Robustness. Inherited from earlier versions of IMITATOR, one can apply the *inverse method* [29] (also called trace preservation [21]) that, given a reference parameter valuation, synthesizes the set of parameter valuations for which the set of "traces" (discrete behaviors, i.e., abstracting time information away) is the same as for this reference valuation.

```
#synth TracePreservation(pTotal = 10, pNeed = 5, pCoffee = 3, max = 3)
```

The result is: $\big(3 \times p_{need} > p_{total} \geq 2 \times p_{need} \land max \in [2, 3)\big) \lor \big(2.1 \times p_{need} > p_{total} \geq 2 \times p_{need} \land max \geq 3\big)$. The synthesized constraint can be seen as a characterization of the *robustness* of the original parameter valuation.

Synthesis Using Patterns. Another way to specify properties is to use a set of predefined observer patterns [6,28]. Observer patterns are translated into observer automata (called reachability testing in [1]), and their correctness reduces to reachability. This procedure is transparent to the user, i.e., (s)he only needs to specify the pattern and IMITATOR takes care of the translation and synthesis. IMITATOR patterns specify the order between actions, extended with (possibly parametric) timing information. The syntax is detailed in the user manual, and the semantics is given in [6].

For example, one can synthesize the set of valuations such that, every time the researcher restarts a new article, (s)he completes it within 5 time units. That is, every occurrence of the *restart* action must be followed within (at most) 5 time units by the *done* action.

```
#synth pattern(everytime restart then eventually done within 5)
```

A part of the valuations set is: $max \geq 6 \wedge 5 - 6 \times p_{coffee} \geq p_{total} \geq 4.7 \times p_{need}$.

A graphical 2D representation projected onto p_{total} and p_{coffee} (setting $p_{need} = 2$ and $max = 3$) is given in Fig. 1b.

Other Algorithms. IMITATOR features a number of additional algorithms, including *i) non-Zeno infinite run synthesis* [27], *ii) behavioral cartography* [16] that partitions the parameter space into *tiles* where the discrete behavior is uniform, or *iii) parametric reachability preservation*, that takes as input a discrete state and a reference valuation, and synthesizes valuations for which this discrete state is reachable iff it is reachable for the reference valuation [25]. The two latter algorithms can be distributed over a cluster, showing interesting results, and can be used to perform reachability synthesis while being faster than the normal reachability synthesis algorithm for some benchmarks [14,15]. Finally, compositional verification for a subclass of IPTAs (a parametric extension of event-recording automata [4]) was proposed in [24].

4 Distribution

IMITATOR is distributed under the terms of the GNU General Public License. Its source code is therefore publicly available, and benefited from several contributors' additions. IMITATOR is available online[2], together with its documentation, and a benchmarks library [26].

IMITATOR depends on several libraries. Notably, the core engine relies on the Parma Polyhedra Library (PPL) [32] for the computation of symbolic states. As a consequence, IMITATOR can be cumbersome to compile. For this reason, standalone binaries are available for all Linux-like systems. A Docker version[3] (made by Jaime Arias) and a prototype Web service[4] are available too.

An extensive user manual, explaining all algorithms and providing users with a full description of the input syntax for models and properties, is available [10].

5 A Selection of Applications

IMITATOR was applied to a variety of both academic and industrial case studies over the last few years. These applications range within several domains, including real-time systems, testing and monitoring, cybersecurity, or hardware verification. One can cite:

[2] https://www.imitator.fr.
[3] https://hub.docker.com/r/imitator/imitator/.
[4] https://imitator.lipn.univ-paris13.fr/.

- the parametric verification of an asynchronous memory circuit by ST-Microelectronics (from a model described in [37]),
- verification of parametric scheduling problems by Astrium Space Transportation [40] and ArianeGroup SAS [13],
- analysis of music scores [38],
- verifying the multi-processor image processing system of an unmanned aerial aircraft with uncertain periods, as a benchmark made public by Thales [46],
- parametric pattern matching and monitoring of logs from the automative industry [20],
- synthesis of timing/cost parameters in attack-fault trees [23,31],
- testing product lines using parametric constraints [44],
- verification of an industrial asynchronous leader election algorithm by Thales using IMITATOR combined with abstractions [18],
- performing parametric opacity analyses for timed automata [30], and
- synthesis of parameter valuations guaranteeing liveness properties for the Bounded Retransmission Protocol [11].

6 Related Tools

HyTech [41] was the first model checker for hybrid systems (a class of formalisms beyond PTAs), including parameters; it is not maintained anymore.

Uppaal [42] is a state-of-the-art tool for modeling and verifying systems modeled as networks of timed automata and extended with variables and data structures; while Uppaal became a major tool for model checking timed automata, it does not support parametric verification, and the use of clocks is restricted to comparing one clock with one constant or with another clock, while IMITATOR allows a liberal syntax based on polyhedra.

RomÉo [43] performs parameter synthesis for parametric time Petri nets with inhibitor arcs [47].

While RomÉo shares similarities with IMITATOR, it does not support (extensions of) timed automata, and notably not multi-rate clocks.

SpaceEx [39] is a tool for verifying hybrid systems. It is not specifically dedicated to parameter synthesis, and mainly targets safety and reachability, in contrast to IMITATOR that proposes multiple synthesis algorithms.

IMITATOR's input syntax also shares some similarities with that of PHAVerLite [33] (a fork of PHAVer and predecessor of SpaceEx, that uses PPLite [34] instead of PPL [32]), coming from the fact that both IMITATOR and PHAVerLite originate from the HyTech syntax.

7 Perspectives

To gain some further speed for models that require less expressiveness (notably no strict inequality nor rational-valued variables), offering to replace PPL [32] with PPLite [34], or using standard 32-bit integers instead of GNU MP rationals is on our agenda.

Acknowledgement. While the author has been the main developer of IMITATOR since 2008, several colleagues brought very valuable enhancements, notably Camille Coti and Sami Evangelista [14] (on distributed algorithms), Nguyễn Hoàng Gia [27] (on non-Zeno algorithms), Vincent Bloemen [12] (on minimal-time synthesis), Laure Petrucci and Jaco van de Pol [11] (on NDFS-based cycle synthesis), and Jaime Arias for multiple practical enhancements. Many thanks to Dylan Marinho for a careful rereading of this paper, and to Stephan Merz for useful suggestions.

References

1. Aceto, L., Bouyer, P., Burgueño, A., Larsen, K.G.: The power of reachability testing for timed automata. TCS **300**(1–3), 411–475 (2003). https://doi.org/10.1016/S0304-3975(02)00334-1
2. Alur, R., et al.: The algorithmic analysis of hybrid systems. TCS **138**(1), 3–34 (1995). https://doi.org/10.1016/0304-3975(94)00202-T
3. Alur, R., Dill, D.L.: A theory of timed automata. TCS **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
4. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: a determinizable class of timed automata. TCS **211**(1–2), 253–273 (1999). https://doi.org/10.1016/S0304-3975(97)00173-4
5. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: Kosaraju, S.R., Johnson, D.S., Aggarwal, A. (eds.) STOC, pp. 592–601. ACM, New York, NY, USA (1993). https://doi.org/10.1145/167088.167242
6. André, É.: Observer patterns for real-time systems. In: Liu, Y., Martin, A. (eds.) ICECCS, pp. 125–134. IEEE Computer Society, July 2013. https://doi.org/10.1109/ICECCS.2013.26
7. André, É.: Parametric deadlock-freeness checking timed automata. In: Sampaio, A., Wang, F. (eds.) ICTAC 2016. LNCS, vol. 9965, pp. 469–478. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46750-4_27
8. André, É.: What's decidable about parametric timed automata? STTT **21**(2), 203–219 (2019). https://doi.org/10.1007/s10009-017-0467-0
9. André, É.: Artifact for IMITATOR 3.0, April 2021. https://doi.org/10.5281/zenodo.4723415
10. André, É.: IMITATOR user manual, January 2021. https://github.com/imitator-model-checker/imitator/releases/download/v3.0.0/IMITATOR-user-manual.pdf
11. André, É., Arias, J., Petrucci, L., Pol, J.: Iterative bounded synthesis for efficient cycle detection in parametric timed automata. In: TACAS 2021. LNCS, vol. 12651, pp. 311–329. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72016-2_17
12. André, É., Bloemen, V., Petrucci, L., van de Pol, J.: Minimal-time synthesis for parametric timed automata. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11428, pp. 211–228. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17465-1_12
13. André, É., Coquard, E., Fribourg, L., Jerray, J., Lesens, D.: Scheduling synthesis for a launcher flight control using parametric stopwatch automata. In: Keller, J., Penczek, W. (eds.) ACSD, pp. 13–22. IEEE (2019). https://doi.org/10.1109/ACSD.2019.00006
14. André, É., Coti, C., Evangelista, S.: Distributed behavioral cartography of timed automata. In: Dongarra, J., Ishikawa, Y., Atsushi, H. (eds.) EuroMPI/ASIA, pp. 109–114. ACM, September 2014. https://doi.org/10.1145/2642769.2642784

15. André, É., Coti, C., Nguyen, H.G.: Enhanced distributed behavioral cartography of parametric timed automata. In: Butler, M., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 319–335. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25423-4_21
16. André, É., Fribourg, L.: Behavioral cartography of timed automata. In: Kučera, A., Potapov, I. (eds.) RP 2010. LNCS, vol. 6227, pp. 76–90. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15349-5_5
17. André, É., Fribourg, L., Kühne, U., Soulat, R.: IMITATOR 2.5: a tool for analyzing robustness in scheduling problems. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 33–36. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32759-9_6
18. André, É., Fribourg, L., Mota, J.-M., Soulat, R.: Verification of an industrial asynchronous leader election algorithm using abstractions and parametric model checking. In: Enea, C., Piskac, R. (eds.) VMCAI 2019. LNCS, vol. 11388, pp. 409–424. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11245-5_19
19. André, É., Fribourg, L., Soulat, R.: Merge and conquer: state merging in parametric timed automata. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 381–396. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02444-8_27
20. André, É., Hasuo, I., Waga, M.: Offline timed pattern matching under uncertainty. In: Lin, A.W., Sun, J. (eds.) ICECCS, pp. 10–20. IEEE Computer Society (2018). https://doi.org/10.1109/ICECCS2018.2018.00010
21. André, É., Lime, D., Markey, N.: Language preservation problems in parametric timed automata. LMCS 16, January 2020. https://doi.org/10.23638/LMCS-16(1:5)2020
22. André, É., Lime, D., Ramparison, M.: Parametric updates in parametric timed automata. In: Pérez, J.A., Yoshida, N. (eds.) FORTE 2019. LNCS, vol. 11535, pp. 39–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21759-4_3
23. André, É., Lime, D., Ramparison, M., Stoelinga, M.: Parametric analyses of attack-fault trees. In: Keller, J., Penczek, W. (eds.) ACSD, pp. 33–42. IEEE (2019). https://doi.org/10.1109/ACSD.2019.00008
24. André, É., Lin, S.-W.: Learning-based compositional parameter synthesis for event-recording automata. In: Bouajjani, A., Silva, A. (eds.) FORTE 2017. LNCS, vol. 10321, pp. 17–32. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60225-7_2
25. André, É., Lipari, G., Nguyen, H.G., Sun, Y.: Reachability preservation based parameter synthesis for timed automata. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 50–65. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17524-9_5
26. André, É., Marinho, D., van de Pol, J.: A benchmarks library for extended timed automata. In: Loulergue, F., Wotawa, F. (eds.) TAP (2021). (to appear)
27. André, É., Nguyen, H.G., Petrucci, L., Sun, J.: Parametric model checking timed automata under non-zenoness assumption. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 35–51. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57288-8_3
28. André, É., Petrucci, L.: Unifying patterns for modelling timed relationships in systems and properties. In: Moldt, D., Rölke, H., Störrle, H. (eds.) PNSE, vol. 1372, pp. 25–40. CEUR-WS, June 2015
29. André, É., Soulat, R.: The Inverse Method. FOCUS Series in Computer Engineering and Information Technology, p. 176, ISTE Ltd and John Wiley & Sons Inc. Hoboken (2013)

30. André, É., Sun, J.: Parametric timed model checking for guaranteeing timed opacity. In: Chen, Y.-F., Cheng, C.-H., Esparza, J. (eds.) ATVA 2019. LNCS, vol. 11781, pp. 115–130. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31784-3_7
31. Arias, J., Budde, C.E., Penczek, W., Petrucci, L., Sidoruk, T., Stoelinga, M.: Hackers vs. Security: attack-defence trees as asynchronous multi-agent systems. In: Lin, S.-W., Hou, Z., Mahony, B. (eds.) ICFEM 2020. LNCS, vol. 12531, pp. 3–19. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63406-3_1
32. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Sci. Comput. Programm. **72**(1–2), 3–21 (2008). https://doi.org/10.1016/j.scico.2007.08.001
33. Becchi, A., Zaffanella, E.: Revisiting polyhedral analysis for hybrid systems. In: Chang, B.-Y.E. (ed.) SAS 2019. LNCS, vol. 11822, pp. 183–202. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32304-2_10
34. Becchi, A., Zaffanella, E.: PPLite: zero-overhead encoding of NNC polyhedra. Inf. Comput. **275**, 104620 (2020). https://doi.org/10.1016/j.ic.2020.104620
35. Beneš, N., Bezděk, P., Larsen, K.G., Srba, J.: Language emptiness of continuous-time parametric timed automata. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 69–81. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47666-6_6
36. Cassez, F., Larsen, K.: The impressive power of stopwatches. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 138–152. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44618-4_12
37. Chevallier, R., Encrenaz-Tiphène, E., Fribourg, L., Xu, W.: Timed verification of the generic architecture of a memory circuit using parametric timed automata. FMSD **34**(1), 59–81 (2009). https://doi.org/10.1007/s10703-008-0061-x
38. Fanchon, L., Jacquemard, F.: Formal timing analysis of mixed music scores. In: ICMC. Michigan Publishing, August 2013
39. Frehse, G., et al.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
40. Fribourg, L., Lesens, D., Moro, P., Soulat, R.: Robustness analysis for scheduling problems using the inverse method. In: Reynolds, M., Terenziani, P., Moszkowski, B. (eds.) TIME, pp. 73–80. IEEE Computer Society Press, September 2012. https://doi.org/10.1109/TIME.2012.10
41. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: a model checker for hybrid systems. STTT **1**(1–2), 110–122 (1997). https://doi.org/10.1007/s100090050008
42. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. STTT **1**(1–2), 134–152 (1997). https://doi.org/10.1007/s100090050010
43. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.-M.: Romeo: a parametric model-checker for petri nets with stopwatches. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 54–57. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00768-2_6
44. Luthmann, L., Gerecht, T., Stephan, A., Bürdek, J., Lochau, M.: Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints. J. Syst. Softw. **149**, 535–553 (2019). https://doi.org/10.1016/j.jss.2018.12.028

45. Nguyen, H.G., Petrucci, L., van de Pol, J.: Layered and collecting NDFS with subsumption for parametric timed automata. In: Lin, A.W., Sun, J. (eds.) ICECCS, pp. 1–9. IEEE Computer Society, December 2018. https://doi.org/10.1109/ICECCS2018.2018.00009
46. Sun, Y., André, É., Lipari, G.: Verification of two real-time systems using parametric timed automata. In: Quinton, S., Vardanega, T. (eds.) WATERS, July 2015
47. Traonouez, L.M., Lime, D., Roux, O.H.: Parametric model-checking of stopwatch Petri nets. J. Univ. Comput. Sci. **15**(17), 3273–3304 (2009). https://doi.org/10.3217/jucs-015-17-3273