Parametric Scheduling with IMITATOR

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

[1]LIPN, CNRS UMR 7030, Université Paris 13, France
[2]ENS Cachan, CNRS, LSV, UMR8643
[3]Universität Bremen, Germany

Abstract. We present here IMITATOR 2.4.

Keywords: Real-Time Systems, Scheduling, Parametric Timed Automata, Parameter Synthesis

⇐ Version avec commentaires ⇒

1 Motivation

⇐ short introduction ⇒

In [3], we proposed the inverse method *IM* for Timed Automata. Different from CEGAR-based methods, this original semi-algorithm for parameter synthesis is based on a "good" parameter valuation π_0 instead of a set of "bad" states. *IM* synthesizes a constraint K_0 on the parameters such that, for all parameter valuation π satisfying K_0, the trace set (i.e., the discrete behavior) of $\mathcal{A}$ under π is the same as for $\mathcal{A}$ under π_0. This preserves in particular linear time properties, and provides the system with a criterion of robustness.

A basic implementation named IMITATOR (for *Inverse Method for Inferring Time AbstracT behaviOR*) has first been proposed, under the form of a Python script calling HYTECH. The tool has then been entirely rewritten in IMITATOR II [2], under the form of a standalone OCaml program making use of the Parma Polyhedra Library (PPL) [5]. A number of case studies containing up to 60 timing parameters could be efficiently verified in the purely timed framework.

Since [2], we extended IMITATOR to deal with parametric scheduling analysis. First, 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 arbitrary clock resets: clocks can 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*. These extensions allow us to consider larger classes of case studies, and in particular scheduling problems.

⇐ Laurent a dit : - le principe de la methode inverse pour les TA a été implémenté de façon basique ([ICTAC09]) puis de façon sophistiquée ([Infinity10])

- la méthode a été étendue pour les HA et implémentée de façon basique ([RP11])
- il s'agit "ici" de décrire une implémentation sophistiquée (intégrant notamment des extensions du merging des TA à la [NFM 12] aux HA)
⇒

2 Architecture and Features

The core of the program is written in OCaml, and interacts with PPL. Exact arithmetics with unbounded precision is used.

IMITATOR takes as input a network of timed automata 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).

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 the following algorithms:

Full reachability analysis Given a model, it computes the set of symbolic reachable states.

Inverse method Given a model and a reference parameter valuation π_0, it computes a constraint on the parameter guaranteeing the same time-abstract behavior as under π_0

Behavioral cartography Given a model and a bounded parameter domain for each parameter valuation, it computes a set of constraints and their corresponding trace sets.

IMITATOR makes use of several algorithmic optimizations. In particular, we implemented a technique that merge any two states sharing the same discrete part (location and value of the discrete variables) and such that the union of their constraint on the analog variables and parameters is convex [4]. This optimization preserves the correctness of all our algorithms; better, the constraint output by the inverse method in that case may be weaker, i.e., covers a larger set of parameter valuations. This optimization behaves particularly well in the framework of scheduling problems, where the state space is drastically reduced.

3 Applications

Source code, binaries, case studies and logs are available in [1].

⇐ TO DO ⇒

Fig. 1. Examples of graphics output by IMITATOR

⇐ **Laurent a dit :** ⇒

- sampled data hybrid systems (cf [RP11])
- test coverage for hybrid systems
- synthesis schedulability regions (cf [Cimatti-Palopoli-Ramadian08], [Astrium EADS])

⇐ **experiences ?** ⇒

⇐ **rapport d'etudes de cas a rediger** ⇒

4 Related Work

One of the first powerful model checkers for analyzing parametric timed automata is HyTech [6]. Unfortunately, it can hardly verify even medium sized examples due to exact arithmetics with limited precision and static composition of automata, quickly leading to memory overflows.

⇐ **to do (citer Romeo)** ⇒

A graphical user interface for the input model is currently under construction, based on a generic platform. ⇐ **utile ?!** ⇒

References

1. Imitator's Web page. `http://www.lsv.ens-cachan.fr/Software/imitator/`.
2. 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.
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, 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.
5. 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.
6. 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.

Appendix

Summary of Experiments

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

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) ⇒