Contributions to parametric timed model checking: Theory and algorithms

Étienne André

UNIVERSITÉ PARIS 13

Mémoire d'habilitation à diriger des recherches

Contributions to parametric timed model checking: Theory and algorithms

Étienne André

Defended on 25th of June 2018, before the committee composed of

Parosh Aziz Abdulla	Uppsala University	Sweden	Reviewer
Patricia Bouyer-Decitre	CNRS	France	Reviewer
Benoît Caillaud	Inria Rennes	France	Examiner
Thao Dang	CNRS, Université Grenoble Alpes	France	Examiner
Kim Guldstrand Larsen	Aalborg University	Denmark	Reviewer
Giuseppe Lipari	Université de Lille	France	Examiner
Tayssir Touili	Université Paris 13	France	Examiner
Emmanuel Viennet	Université Paris 13	France	President

This manuscript is mostly based on joint works with:

Thomas Chatain	ENS Paris-Saclay	France
Camille Coti	Université Paris 13	France
Benoît Delahaye	Université de Nantes	France
Sami Evangelista	Université Paris 13	France
Laurent Fribourg	CNRS, ENS Paris-Saclay	France
Michał Knapik	Institute of Computer Science, PAS	Poland
Didier Lime	École Centrale de Nantes	France
Lin Shang-Wei	Nanyang Technological University	Singapore
Giuseppe Lipari	Université de Lille	France
Nicolas Markey	CNRS, Univ. Rennes	France
Nguyễn Hoàng Gia	Université Paris 13	France
Wojciech Penczek	ICS PAS, Siedlce University	Poland
Laure Petrucci	Université Paris 13	France
Mathias Ramparison	Université Paris 13	France
César Rodríguez	Diffblue Ltd, Oxford	England
Olivier H. Roux	École Centrale de Nantes	France
Romain Soulat	Thales R&D	France
Sun Jun	Singapore University of Technology and Design	Singapore
Sun Youcheng	University of Oxford	England

Contents

1 Introduction **6**

2 Preliminaries **11**
- 2.1 Parametric timed automata . . . 11
  - 2.1.1 Clocks, Parameters and Constraints . . . 11
  - 2.1.2 Parametric Timed Automata . . . 12
  - 2.1.3 Subclasses of PTAs . . . 16
- 2.2 Decision and computation problems . . . 17
  - 2.2.1 Decision problems . . . 17
  - 2.2.2 Computation problem . . . 19
- 2.3 A semi-algorithm for reachability synthesis . . . 19
- 2.4 A semi-algorithm for trace set preservation synthesis . . . 20

3 Decidability and expressiveness of parametric timed automata **22**
- 3.1 Exploring the jungle of decidability results . . . 22
  - 3.1.1 Almost everything is undecidable for simple PTAs . . . 22
  - 3.1.2 Bounding the numbers of clocks and parameters . . . 24
  - 3.1.3 L/U-PTAs . . . 25
- 3.2 Expressiveness of parametric timed automata . . . 26
  - 3.2.1 A new subclass: Integer-point parametric timed automata . . . 26
  - 3.2.2 Defining the expressiveness of parametric timed automata . . . 28
  - 3.2.3 Comparison of the expressiveness of subclasses of PTAs . . . 29
- 3.3 Decidability of parametric timed automata . . . 32
  - 3.3.1 EF-emptiness . . . 32
  - 3.3.2 AF-emptiness . . . 35
  - 3.3.3 EG-emptiness . . . 36
  - 3.3.4 AG-emptiness . . . 37
  - 3.3.5 Nesting quantifiers . . . 38
  - 3.3.6 Cycle-existence emptiness . . . 38
  - 3.3.7 Deadlock-freeness emptiness and synthesis . . . 39
  - 3.3.8 Language and trace preservation . . . 42
  - 3.3.9 The one-clock case . . . 44
  - 3.3.10 Summary of decision problems . . . 44
- 3.4 Perspectives . . . 46

4 Efficient verification **47**
4.1 Convex state merging . . . 48
4.2 Dynamic clocks elimination . . . 49
4.3 Guaranteeing termination with the integer hull . . . 51
4.3.1 Context and objective . . . 51
4.3.2 A parametric extrapolation . . . 52
4.3.3 Ensuring termination of parameter synthesis . . . 53
4.3.4 Implementation . . . 55
4.4 Towards distributed parameter synthesis . . . 55
4.4.1 The behavioral cartography . . . 56
4.4.2 Distribution policies . . . 57
4.5 EF-synthesis using reachability preservation . . . 60
4.5.1 Reachability preservation . . . 60
4.5.2 EF-synthesis . . . 62
4.6 Compositional synthesis for parametric event-recording automata . . . 63
4.6.1 Parametric event-recording automata . . . 63
4.6.2 Learning event-recording automata . . . 65
4.6.3 Compositional verification of event-recording automata . . . 66
4.6.4 Compositional parameter synthesis . . . 67
4.7 Perspectives . . . 68

5 Synthesis algorithms **70**
5.1 Parameter synthesis and robustness . . . 70
5.1.1 Varying the definition of robustness . . . 70
5.1.2 Precise robustness in time Petri nets . . . 72
5.1.3 Robustness and partial orders . . . 75
5.2 Non-Zeno synthesis . . . 78
5.2.1 CUB-parametric timed automata . . . 79
5.2.2 Non-Zeno synthesis . . . 80
5.3 Combining timing parameters with action parameters . . . 80
5.4 Parameter synthesis in probabilistic models . . . 82
5.4.1 Consistency in interval probabilistic timed automata . . . 84
5.4.2 Consistency synthesis in parametric interval probabilistic timed automata . . . 84
5.5 Perspectives . . . 85

6 Application to parametric schedulability **87**
6.1 Parametric schedulability analysis . . . 87
6.1.1 Schedulability analysis using parametric stopwatch automata . . . 88
6.1.2 Experiments and comparison . . . 89
6.2 Parametric task automata: A unified formalism for uniprocessor schedulability . . . 91
6.2.1 Decidability and undecidability . . . 92
6.2.2 Synthesis . . . 93
6.3 The Thales FMTV challenge . . . 94
6.3.1 Challenge description . . . 94
6.3.2 Solution using IMITATOR . . . 95
6.4 Perspectives . . . 97

7 Conclusion **98**

Acknowledgements

I would like to warmly thank Parosh Abdulla, Patricia Bouyer and Kim Larsen for doing me the great honor of reviewing my habilitation. I would also like to thank Benoît Caillaud, Thao Dang, Giuseppe Lipari and Tayssir Touili for accepting to be part of the jury, and Emmanuel Viennet for being the president.

This habilitation would certainly not have been possible without the colleagues who collaborated on these works. I appreciated not only the scientific collaboration, but also the excellent atmosphere in which these joint works were carried out.

Je remercie le CNRS pour m'avoir offert l'opportunité d'une délégation au sein de l'IRCCyN (désormais LS2N) en 2015–2016, laquelle a permis des travaux de recherche fructueux, et de confirmer une collaboration soutenue avec mes collègues nantais, notamment Benoît Delahaye, Didier Lime et Olivier Roux. Une partie de ce manuscrit provient directement des travaux entrepris au cours de ma délégation.

L'Agence Nationale de la Recherche a directement contribué à cette HDR via le soutien du projet ANR PACS (*Parametric Analyses of Timed Systems*) 2014–2019 dont je suis porteur. La plupart des résultats présentés ici ont bénéficié du cadre tant financier que scientifique de ce projet.

Merci à Christophe Fouqueré et Frédérique Bassino pour m'avoir guidé dans la jungle administrative tridécaparisienne d'une soutenance d'HDR, et à Christine Choppy pour ses conseils.

Je remercie chaleureusement l'équipe systèmes temps-réel de feu l'IRCCyN pour leur très sympathique accueil d'un « maître de conférences CNRS » (sic), pour avoir généreusement accepté d'avancer certains jours leur pause-déjeuner dès 13h15, pour les soirées Sergio Leone et Quentin Tarantino dans un cinéma privé de Carquefou (auquel certaines salles parisiennes n'ont rien à envier quant à la taille de l'écran), pour la carte cantine magique où plus l'on mange et moins on paie, pour les pauses café, pour les (presque) soirées raclette, bref pour avoir créé un climat aussi chaleureux que le climat nantais n'est humide.

I would like to thank Dong Jin-Song for hiring me as a post-doc with the National University of Singapore in 2010–2011 and for offering me various interesting opportunities since then.

Un grand merci à Laurent Fribourg pour avoir poursuivi un service après-vente plusieurs années après la fin de ma thèse, par ses conseils toujours judicieux et sa bonne humeur. Et un remerciement tout particulier à Alain Finkel et Serge Haddad sans qui ce mémoire d'habilitation n'aurait (peut-être) jamais vu le jour.

Enfin, merci à mes parents (qui m'ont notamment offert gîte et couvert pendant une partie de la rédaction de ce mémoire) et à mon super-frérot.

Chapter 1

Introduction

Context

Since two decades, the number of embedded systems has risen dramatically. Such systems are critical whenever a failure can have dramatic consequences, such as huge financial costs or even possible loss of human lives. In addition, with the rise of the Internet of Things in a very near future, the number of critical devices in smart homes, smart cities (including autonomous cars), will also increase dramatically.

The correctness of critical embedded systems must be asserted before execution. This can be achieved using formal methods, that include mainly model-based testing, theorem proving or model checking. My contributions in this manuscript focus on model checking. Model checking allows to formally assess the correctness of a system using a model (a mathematical abstraction) and a property. Three of the main scientists behind this paradigm (Edmund M. Clarke, Allen Emerson and Joseph Sifakis) were awarded the ACM's Turing Award in 2007. In the original setting, model checking can take as input a finite state automaton, and a property, typically expressed in a temporal logic such as LTL or CTL. A simple example is depicted in Figure 1.1.

This simple setting, although used in powerful tools with industrial successes, falls short when the designers need to model and verify *quantitative* aspects such as time, energy, probabilities, etc. Timed automata [AD94] allow for modeling and verifying systems where concurrency is combined with critical timing aspects. This seminal work [AD94] was awarded the CAV award in 2008, and numerous works extended this formalism since then.

However, despite some success, (timed) model checking can be seen as slightly disappointing. Two major reasons are the binary response to properties satisfaction, which may not be informative enough,

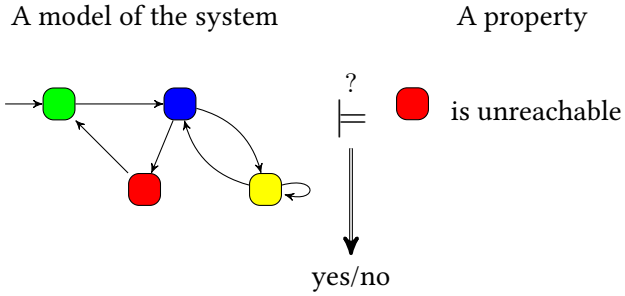


Figure 1.1 – Model checking

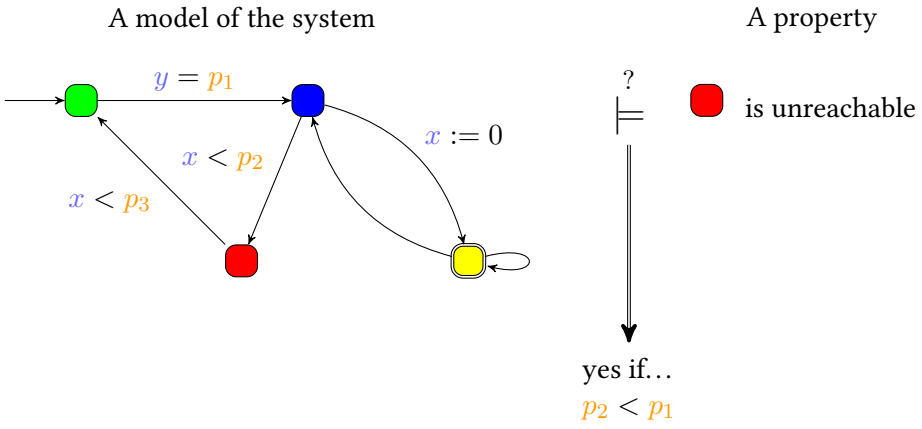


Figure 1.2 – Parametric timed model checking

and the insufficient abstraction to cater for tuning and scalability of systems. Parameters offer a higher level of abstraction, by allowing unknown constants in a model. In model checking timed systems, parameters can model unknown *timing* constants. The interest is two fold: first, it becomes possible to verify a system at an earlier design stage, when all timing constants are not yet known with full certainty. And, second, it allows designers to cope with uncertainty even at runtime: some timings constants (e. g., periods of a real-time system) might be known with only some precision, and parameters can model this imprecision. In fact, the problem with a binary answer of Figure 1.1 becomes richer when augmented with parameters: instead of answering "yes" or "no", parametric model checking *synthesizes* values for which the system meets its specification. This is depicted in Figure 1.2.

Parametric timed automata [AHV93] are an extension of timed automata where timing constants can become unknown, i. e., *parameters*. They represent a particularly expressive formalism: in fact, its expressiveness is Turing-complete and all non-trivial problems are undecidable.

Parametric timed automata suffer from negative results when coming to decidability, but they remain an extremely powerful formalism. They can help to address robustness (in the sense of possibly infinitesimal variations of timing constants [BMS13]), can model and verify systems with uncertain constants, and can synthesize suitable (possibly unknown) valuations so that the system meets its specification.

In addition, several recent decidability results for subclasses of parametric timed automata (e. g., [BL09; BO14; JLR15; Ben+15]) made this formalism more promising, while new algorithmic and heuristic techniques (e. g., [KP12; JLR15; Aşt+16]) made the parametric verification for some classes of problems more scalable, more complete, or more often terminating.

Verification with parametric timed automata had concrete outcomes in various areas, with verification of case studies such as the root contention protocol [Hun+02], Philip's bounded retransmission protocol [Hun+02]), a 4-phase handshake protocol [KP12], the alternating bit protocol [JLR15], an asynchronous circuit commercialized by ST-Microelectronics [Che+09], (non-preemptive) schedulability problems [JLR15], a distributed prospective architecture for the flight control system of the next generation of spacecrafts designed at ASTRIUM Space Transportation [Fri+12], and even analysis of music scores [FJ13].

In the past decade, I contributes to the state-of-the-art of parametric verification on three directions of research:

1. exhibit decidable subclasses, and evaluate their complexity;

2. design semi-algorithms (procedures that may not terminate, but if they do, the result is correct), and perform experiments to evaluate how often they terminate;

3. design algorithms with an approximated result (i. e., that may lose the completeness).

I address all three directions in this manuscript. In addition, I focused specifically on parametric schedulability analysis (notably for real-time systems), and solved (with two of my colleagues) an industrial challenge proposed by Thales, using parametric timed automata.

Finally note that abstraction and uncertainty can also be encoded using *probabilities*: this is also a setting I used (together with timing parameters) in one of my contributions.

Content of this manuscript

Chapter 2 recalls the necessary material, mainly parametric timed automata and decision problems.

Chapter 3 reports on contributions concerning the decidability of parametric timed automata and their subclasses. I start by performing a survey of (nearly) all known decidability results for parametric timed automata, their subclasses or variants. I then present new decidability results, and exhibit several novel decidable subclasses, namely (variants of) lower-bound/upper-bound parametric timed automata (initially defined in [Hun+02]) and integer-point parametric timed automata [ALR16a].

This chapter is mainly based on [And18; ALR16b; AM15; ALR16a; And16; AL17a; ALR18]. Recurrent and main collaborators on this topic are Didier Lime, Olivier H. Roux and Nicolas Markey. The beginning of Mathias Ramparison's PhD thesis (who started in September 2016) is also integrated [ALR18].

Chapter 4 summarizes contributions related to techniques to speed up parameter synthesis in practice, independently of the decidability. That is, even in the large class of parametric timed automata, we design optimizations that may be exact, or approximated—in the latter case, termination is ensured. The first two techniques are techniques initially proposed for timed automata, that we extended to the parametric setting, i. e., convex state merging (together with Romain Soulat and Laurent Fribourg), and dynamic clock elimination. Then, the integer hull (in collaboration with Didier Lime and Olivier Roux) is a technique specifically designed for parameter synthesis (together with a new notion of parametric extrapolation), that ensures termination while synthesizing at least all integer valuations (in a bounded domain).

In a different line of work, we show that parameter synthesis can be sped up using distributed verification (with Camille Coti, Sami Evangelista and Nguyễn Hoàng Gia). We then report on an algorithm that makes the distributed parameter synthesis more efficient (with Giuseppe Lipari and Sun Youcheng), based on the notion of preservation of reachability (a given location is reachable for all synthesized valuations iff it is reachable in a given non-parametric model). Then, using a learning algorithm for a subclass of non-parametric timed automata (namely event-recording automata [AFH99]), we can infer an abstraction of a system, that is used in compositional verification (with Lin Shang-Wei). Combining several of this works together, we can finally design a semi-algorithm for compositional parameter synthesis for parametric even-recording automata (again with Lin Shang-Wei).

This chapter is mainly based on [AFS13a; And13a; ACE14; ACN15; And+15; Lin+14; AL17b; AL18].

The results of Nguyễn Hoàng Gia's PhD thesis are partially integrated in this chapter [ACN15; And+15].

Chapter 5 addresses the problem of synthesis in practice, that is to design (efficient) algorithms to solve concrete problems. Notably, I am interested in robustness in parametric timed automata and parametric time Petri nets [TLR09] (with Thomas Chatain and César Rodríguez), a formalism that resembles parametric timed automata, but that allows for an explicit notion of concurrency. I also addressed the problem of parameter synthesis under the non-Zeno assumption (with Nguyễn Hoàng Gia, Laure Petrucci and Sun Jun), that rules out Zeno runs, i. e., containing an infinite number of actions in a finite time. Finally, I consider two extensions of parametric timed automata. First, extended with *action parameters*, i. e., when part of the alphabet can be enabled or disabled once for all, in the framework of a joint project with Michał Knapik and Wojciech Penczek. And second, extended with *probabilistic intervals*, where the problem becomes to

synthesize timing parameter valuations for which a probabilistic model admits a consistent implementation; this latter work was done with Benoît Delahaye who brought his expertise in probabilistic interval Markov chains.

This chapter is mainly based on [AS11; APP13; ACR17; And+17b; And+16; AD16].

The results of Nguyễn Hoàng Gia's PhD thesis (co-supervised by Laure Petrucci) are partially integrated in this chapter [And+17b].

Chapter 6 reviews contributions that focus specifically on real-time systems. First, parametric schedulability analysis can be performed using parametric timed automata (extended with additional features such as stopwatches), and can cope with real-time systems where some timings constants (typically periods or deadlines) may be unknown. With Laurent Fribourg, Giuseppe Lipari and, Sun Youcheng, we performed a comparison with analytical methods that show that our approach is generally slower, but on the other hand more complete, and can deal with non-integer valuations. We also review parametric task automata, a parametric extension of task automata [NWY99; Fer+07] that I recently proposed. Finally, as a more practical achievement, we report on a challenge by Thales on a aerial video system with uncertain periods, that we were able to solve using IMITATOR (with Giuseppe Lipari and Sun Youcheng).

This chapter is mainly based on [Sun+13b; And17; ALS15].

Most of the techniques summarized in this manuscript have been implemented (mainly in my tool IMITATOR [And+12]). This is mentioned after each line of work (when applicable).

Local perspectives are given at the end of each chapter. In Chapter 7, I summarize contributions and give general perspectives.

Other recent contributions

This manuscript synthesizes a selection of my research works since my PhD thesis in December 2010. I briefly mention in the following some contributions that are not included in this manuscript, because they lie aside from the main research directions of this manuscript. The reader is referred to my Web page[1] for a complete and up-to-date list of publications.

System specification This manuscript mainly focuses on system verification, without much about the specification phase.

In [And13b], I proposed a set of non-compositional patterns for the verification of timed systems, coming from frequently used properties. The verification of these patterns reduces to reachability; a syntax is defined, and implemented in IMITATOR. We extended these patterns with Laure Petrucci in [AP15], adding a limited dose of compositionality, while addressing both the specification of systems and of formulas. We showed that these patterns can encode some well-known patterns of the literature.

In [And+14], I proposed a parametric extension of the process algebra Stateful timed CSP [Sun+13a], itself a timed extension of Hoare's communicating sequential processes [Hoa85]. When compared to other formalisms such as (parametric) timed automata, (parametric) stateful timed CSP has the advantage of giving the designer the ability to specify hierarchical systems. I implemented this language into a prototype tool ("PSyHCoS") with some synthesis algorithms [And+13].

In a different line of work, in particular around Mahdi Benmoussa's PhD thesis (co-supervised with Christine Choppy), we equipped a part of the Unified Modeling Language (UML) syntax [OMG12] with a formal semantics. In [ACK12; ABC14a; ABC14b; ABC16] we translated most syntactic constructs of the UML state machines into colored Petri nets. In [ACR13; ACN14], with Christine Choppy, Thierry Noulamo

[1] https://lipn.univ-paris13.fr/~andre/publications.php

and Gianna Reggio, we formalized (and extended) a subset of the syntax of UML activity diagrams, using a translation into colored Petri nets.

Web services under uncertainty In [Tan+13], I proposed with Tan Tian Huat a method to synthesize suitable response times of Web services; our method is inspired by state-of-the-art techniques for parameter synthesis in formalisms such as parametric timed automata. We defined an extension of BPEL (Business Process Execution Language) and proposed a symbolic semantics and synthesis algorithms. We then proposed extensions with genetic algorithms [Tan+14] and probabilistic refinement [Tan+16].

Exploration order In [ANP17], we studied several exploration orders of the symbolic states of parametric timed automata, so as to gain efficiency while performing parameter synthesis. This work was carried out during Nguyễn Hoàng Gia's PhD thesis, that I co-supervise with Laure Petrucci.

Machine learning In [Li+17], we used active learning and classification techniques to "guess" potential parameter constraints after repeated calls to the non-parametric model checker Uppaal. Only after a constraint is guessed, IMITATOR is invoked to verify this constraint, leading to a dramatic gain in computation time when compared to a purely parametric analysis. This work was carried out during Li Jiaying's PhD thesis, a PhD student of Sun Jun.

Alternating temporal logic Finally, with Wojtek Jamroga, Michał Knapik, Wojciech Penczek and Laure Petrucci, we recently started in [And+17a] to consider a discrete-time extension of the Alternating temporal logic (ATL), initially defined in [LMO06]. We compared several existing semantics (in addition to a new *counting* semantics), and compared their expressiveness. A natural future extension will be to introduce timing parameters, in the model and/or in the formula.

Chapter 2

Preliminaries

We first recall the formalism of parametric timed automata (Section 2.1). We introduce decision and computation problems (Section 2.2), and recall a semi-algorithm for reachability synthesis (Section 2.3).

2.1 Parametric timed automata

2.1.1 Clocks, Parameters and Constraints

Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}_+$ and $\mathbb{R}_+$ denote the sets of non-negative integers, integers, non-negative rational numbers and non-negative real numbers respectively.

Throughout this manuscript, we assume a set $X = \{x_1, \ldots, x_H\}$ of *clocks*, i. e., real-valued variables that evolve at the same rate. A clock valuation is a function $w : X \to \mathbb{R}_+$. We identify a clock valuation w with the point $(w(x_1), \ldots, w(x_H))$ of $\mathbb{R}_+^H$. We write $\vec{0}$ for the clock valuation that assigns 0 to all clocks. Given $d \in \mathbb{R}_+$, $w + d$ denotes the valuation such that $(w + d)(x) = w(x) + d$, for all $x \in X$. Given $R \subseteq X$, we define the *reset* of a valuation w, denoted by $[w]_R$, as follows: $[w]_R(x) = 0$ if $x \in R$, and $[w]_R(x) = w(x)$ otherwise.

We assume a set $P = \{p_1, \ldots, p_M\}$ of *parameters*, i. e., unknown constants. A parameter *valuation* v is a function $v : P \to \mathbb{Q}_+$. We identify a valuation v with the point $(v(p_1), \ldots, v(p_M))$ of $\mathbb{Q}_+^M$. An *integer* parameter valuation is a valuation $v : P \to \mathbb{N}$.

In the following, we assume $\bowtie \in \{<, \leq, \geq, >\}$. A linear term over $X \cup P$ is of the form $\sum_{1 \leq i \leq H} \alpha_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + d$, with $x_i \in X$, $p_j \in P$, and $\alpha_i, \beta_j, d \in \mathbb{Z}$. Throughout this paper, *plt* denotes a parametric linear term over P, that is a linear term without clocks (i. e., $\alpha_i = 0$ for all $1 \leq i \leq H$. A *constraint* C (i. e., a convex polyhedron) over $X \cup P$ is a conjunction of inequalities of the form $lt \bowtie 0$, where lt is a linear term. A *simple inequality* is of the form $x \bowtie p$ or $x \bowtie d$ (with $d \in \mathbb{Q}_+$), and a *simple constraint* is a conjunction of simple inequalities. A *diagonal inequality* is of the form $x_i - x_j \bowtie plt$, and a *diagonal constraint* is a conjunction of diagonal inequalities.

Given a parameter valuation v, $v(C)$ denotes the constraint over X obtained by replacing each parameter p in C with $v(p)$. Likewise, given a clock valuation w, $w(v(C))$ denotes the expression obtained by replacing each clock x in $v(C)$ with $w(x)$. We say that v *satisfies* C, denoted by $v \models C$, if the set of clock valuations satisfying $v(C)$ is nonempty. Given a parameter valuation v and a clock valuation w, we denote by $w|v$ the valuation over $X \cup P$ such that for all clocks x, $w|v(x) = w(x)$ and for all parameters p, $w|v(p) = v(p)$. We use the notation $w|v \models C$ to indicate that $w(v(C))$ evaluates to true. We say that C is *satisfiable* if $\exists w, v$ s.t. $w|v \models C$. An *integer point* is $w|v$, where w is an integer clock valuation, and v is an integer parameter valuation.

We define the *time elapsing* of C, denoted by $C^{\nearrow}$, as the constraint over X and P obtained from C by delaying all clocks by an arbitrary amount of time. That is,

$$w'|v \models C^{\nearrow} \text{ iff } \exists w : X \to \mathbb{R}_+, \exists d \in \mathbb{R}_+ \text{ s.t. } w'|v \models C \wedge w' = w + d.$$

We define the *past* of C, denoted by $C^{\swarrow}$, as the constraint over X and P obtained from C by letting time pass backward by an arbitrary amount of time (see e. g., [JLR15]). That is,

$$w'|v \models C^{\swarrow} \text{ iff } \exists w : X \to \mathbb{R}_+, \exists d \in \mathbb{R}_+ \text{ s.t. } w'|v \models C \wedge w' = w - d \wedge \forall x \in X : w'(x) \geq 0.$$

Given $R \subseteq X$, we define the *reset* of C, denoted by $[C]_R$, as the constraint obtained from C by resetting the clocks in R, and keeping the other clocks unchanged. We denote by $C{\downarrow_P}$ the projection of C onto P, i. e., obtained by eliminating the clock variables (e. g., using Fourier-Motzkin [Sch86]).

A *parametric guard* g is a constraint over $X \cup P$ defined by inequalities of the form $x \bowtie plt$.

$\top$ denotes the constraint over P containing all parameter valuations, while $\bot$ denotes the constraint containing no valuation.

2.1.2 Parametric Timed Automata

Syntax

Definition 2.1 (parametric timed automaton [AHV93])**.** A *parametric timed automaton* (PTA) is a tuple $\mathcal{A} = (\Sigma, L, l_0, F, X, P, I, E)$, where:

1. Σ is a finite set of actions,
2. L is a finite set of locations,
3. $l_0 \in L$ is the initial location,
4. $F \subseteq L$ is a set of final or accepting locations,
5. X is a finite set of clocks,
6. P is a finite set of parameters,
7. I is the invariant, assigning to every $l \in L$ a parametric guard $I(l)$,
8. E is a finite set of edges $e = (l, g, \sigma, R, l')$ where $l, l' \in L$ are the source and target locations, $\sigma \in \Sigma$, $R \subseteq X$ is a set of clocks to be reset, and g is a parametric guard called the transition guard.

Given a parameter valuation v, we denote by $v(\mathcal{A})$ the non-parametric timed automaton where all occurrences of a parameter p_i have been replaced by $v(p_i)$. If $v(\mathcal{A})$ is such that all constants in guards and resets are integers, then $v(\mathcal{A})$ is a *timed automaton* [AD94]. In the following, we may refer to as a timed automaton any structure $v(\mathcal{A})$, by assuming a rescaling of the constants: by multiplying all constants in $v(\mathcal{A})$ by their least common denominator, we obtain an equivalent (integer-valued) timed automaton.

We say that a PTA is *deterministic* if, for any $l \in L$, for any $\sigma \in \Sigma$, there exists at most one edge $(l, g, \sigma, R, l') \in E$, for some g, R, l'. (Note that it differs from a rather common definition of determinism for TAs, that allows two or more outgoing transitions with the same action label provided that the corresponding guards are pairwise disjoint.)

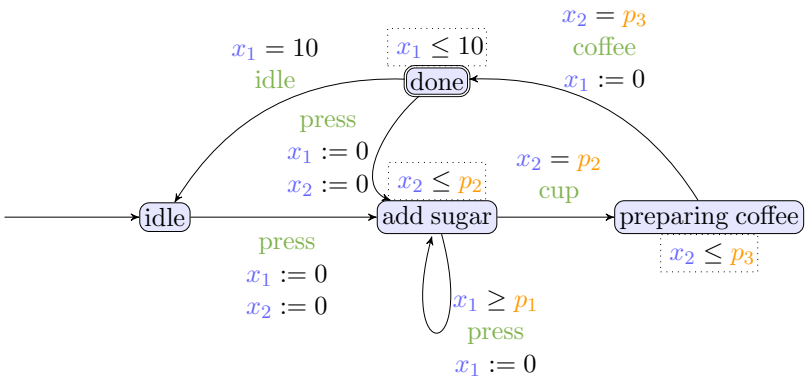


Figure 2.1 – A coffee machine modeled using a PTA

A clock is said to be a *parametric clock* if it is compared with at least one parameter in at least one guard or invariant; otherwise, it is a *non-parametric clock*. This notion is central when studying the decidability of problems for PTAs with few clocks and parameters.

Example 2.1. Consider the coffee machine in Figure 2.1, modeled using a PTA with 4 locations, 2 clocks (x_1 and x_2) and 3 parameters (p_1, p_2, p_3). Invariants are boxed. The only accepting location (with a double border) is done. Both clocks x_1 and x_2 are parametric clocks. Observe that all guards and invariants are simple constraints.

The machine can initially idle for an arbitrarily long time. Then, whenever the user presses the (unique) button (action press), the PTA enters location "add sugar", resetting both clocks. The machine can remain in this location as long as the invariant ($x_2 \leq p_2$) is satisfied; there, the user can add a dose of sugar by pressing the button (action press), provided the guard ($x_1 \geq p_1$) is satisfied, which resets x_1. That is, the user cannot press twice the button (and hence add two doses of sugar) within a time less than p_1. Then, p_2 time units after the machine left the idle mode, a cup is delivered (action cup), and the coffee is being prepared; eventually, p_2 time units after the machine left the idle mode, the coffee (action coffee) is delivered. Then, after 10 time units, the machine returns to the idle mode—unless a user again requests a coffee by pressing the button.

Concrete Semantics

Definition 2.2 (Concrete semantics of a TA)**.** Given a PTA $\mathcal{A} = (\Sigma, L, l_0, F, X, P, I, E)$, and a parameter valuation v, the concrete semantics of $v(\mathcal{A})$ is given by the timed transition system $(S, s_0, \to)$, with

- $S = \{(l, w) \in L \times \mathbb{R}_+^H \mid w|v \models I(l)\}$,
- $s_0 = (l_0, \vec{0})$, and
- $\to$ consists of the discrete and (continuous) delay transition relations:
  - discrete transitions: $(l, w) \xrightarrow{e} (l', w')$, if $(l, w), (l', w') \in S$, there exists $e = (l, g, \sigma, R, l') \in E$, $w' = [w]_R$, and $w|v \models g$.
  - delay transitions: $(l, w) \xrightarrow{d} (l, w + d)$, with $d \in \mathbb{R}_+$, if $\forall d' \in [0, d], (l, w + d') \in S$.

Moreover we write $(l, w) \overset{e}{\mapsto} (l', w')$ for a combination of a delay and discrete transition where

$((l,w),e,(l',w')) \in \mapsto$ if $\exists d, w'' : (l,w) \xrightarrow{d} (l,w'') \xrightarrow{e} (l',w')$.

Given a TA $v(\mathcal{A})$ with concrete semantics $(S, s_0, \to)$, we refer to the states of S as the *concrete states* of $v(\mathcal{A})$. A (concrete) *run* of $v(\mathcal{A})$ is a possibly infinite alternating sequence of concrete states of $v(\mathcal{A})$ and edges starting from the initial concrete state s_0 of the form $s_0 \overset{e_0}{\mapsto} s_1 \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} s_m \overset{e_m}{\mapsto} \cdots$, such that for all $i = 0, 1, \ldots$, $e_i \in E$, and $(s_i, e_i, s_{i+1}) \in \mapsto$. Given a state $s = (l, w)$, we say that s is reachable (or that $v(\mathcal{A})$ reaches s) if s belongs to a run of $v(\mathcal{A})$. By extension, we say that l is reachable in $v(\mathcal{A})$, if there exists a state (l, w) that is reachable. By extension, given a set of locations $T \subseteq L$ (T stands for "target"), we say that T is reachable in $v(\mathcal{A})$, if there exists a location $l \in T$ that is reachable in $v(\mathcal{A})$. We denote by $\mathsf{ReachLocs}(v(\mathcal{A})$ the set of reachable locations, i. e., the set of locations belonging to a run of $v(\mathcal{A})$. Given a set of locations $T \subseteq L$, we say that a run *stays* in T if all of its states (l, w) are such that $l \in T$.

A *maximal run* is a run that is either infinite (i. e., contains an infinite number of discrete transitions), or that cannot be extended by a discrete transition. A maximal run is *deadlocked* if it is finite, i. e., contains a finite number of discrete transitions. By extension, we say that a TA is deadlocked if it contains at least one deadlocked run.

Example 2.2. Consider again the PTA modeling a coffee machine in Figure 2.1. Let v be the parameter valuation such that $v(p_1) = 1$, $v(p_2) = 5$ and $v(p_3) = 8$.

Given a clock valuation w, we denote it by $(w(x_1), w(x_2))$. For example, $(0, 4.2)$ denotes that $w(x_1) = 0$ and $w(x_2) = 4.2$.

The following sequence is a concrete run of $v(\mathcal{A})$.

$(\text{idle}, (0,0)) \overset{\text{press}}{\mapsto} (\text{addsugar}, (0,0)) \overset{\text{press}}{\mapsto} (\text{addsugar}, (0, 1.78)) \overset{\text{press}}{\mapsto} (\text{addsugar}, (0, 4.2)) \overset{\text{cup}}{\mapsto}$
$(\text{preparingcoffee}, (0,5)) \overset{\text{coffee}}{\mapsto} (\text{done}, (3,8)) \overset{\text{press}}{\mapsto} (\text{sugar}, (0,0))$

As an abuse of notation, we write above each arrow the action name (instead of the edge), as edges are unnamed in Figure 2.1.

This concrete run is not maximal (it could be extended).

Language of timed automata

Let $(l_0, w_0) \overset{e_0}{\mapsto} (l_1, w_1) \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} (l_m, w_m) \overset{e_m}{\mapsto} \cdots$ be a (finite or infinite) run of a TA $v(\mathcal{A})$. The associated *untimed word* is $\sigma_0\sigma_1 \cdots \sigma_m \cdots$, where σ_i is the action of edge e_i, for all $i \geq 0$; the associated *trace*[1] is $l_0\sigma_0 l_1 \sigma_1 l_2 \cdots \sigma_m l_{m+1} \cdots$

Given a run $(l_0, w_0) \overset{e_0}{\mapsto} (l_1, w_1) \overset{e_1}{\mapsto} \cdots \overset{e_{m-1}}{\mapsto} (l_m, w_m)$, we say that this run is *accepting* if $l_m \in F$.

Definition 2.3 (untimed language of a TA)**.** Given a PTA $\mathcal{A} = (\Sigma, L, l_0, F, X, P, I, E)$, and a parameter valuation v, the *untimed language* of $v(\mathcal{A})$, denoted by $\mathsf{UL}(v(\mathcal{A}))$, is the set of untimed words associated with all accepting runs of $v(\mathcal{A})$.

Definition 2.4 (trace set of a TA)**.** Given a PTA $\mathcal{A} = (\Sigma, L, l_0, F, X, P, I, E)$, and a parameter valuation v, the *trace set* of $v(\mathcal{A})$, denoted by $\mathsf{Traces}(v(\mathcal{A}))$, is the set of traces associated with all accepting runs of $v(\mathcal{A})$.

[1] This is a nonstandard definition of traces (compared to e. g., [Gla90]), but we keep this term as it is used in, e. g., [And+09b; AM15].

Example 2.3. Consider again the PTA $\mathcal{A}$ modeling a coffee machine in Figure 2.1. Let v be the parameter valuation such that $v(p_1) = 1$, $v(p_2) = 5$ and $v(p_3) = 8$.

The untimed language of $v(\mathcal{A})$ can be described as follows:

$$\text{press}^{[1..6]}\ \text{cup coffee}\big(\text{idle}^{?}\ \text{press}^{[1..6]}\ \text{cup coffee}\big)^{*}$$

where $\sigma^{[a,b]}$, $\sigma^{?}$, σ^{*} denote between a and b occurrences, zero or one occurrence, and zero or more occurrence(s) of σ, respectively.

The trace set of $v(\mathcal{A})$ can be described as follows:

$$\text{idle (press add sugar)}^{[1..6]}\ \text{cup preparing coffee coffee done}$$
$$\big(\text{(idle idle)}^{?}\ \text{(press add sugar)}^{[1..6]}\ \text{cup preparing coffee coffee done}\big)^{*}$$

Symbolic semantics

Let us now recall the symbolic semantics of PTAs (see e. g., [And+09b]).

Definition 2.5 (Symbolic state)**.** A symbolic state is a pair (l, C) where $l \in L$ is a location, and C its associated parametric zone.

Definition 2.6 (Symbolic semantics)**.** Given a PTA $\mathcal{A} = (\Sigma, L, l_0, F, X, P, I, E)$, the symbolic semantics of $\mathcal{A}$ is the labeled transition system called *parametric zone graph* $\mathcal{PZG} = (E, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$, with

- $\mathbf{S} = \{(l, C) \mid C \subseteq I(l)\}$,
- $\mathbf{s}_0 = \big(l_0, (\bigwedge_{1 \leq i \leq H} x_i = 0)^{\nearrow} \wedge I(l_0)\big)$, and
- $\big((l, C), e, (l', C')\big) \in \Rightarrow$ if $e = (l, g, \sigma, R, l')$ and

$$C' = \big([(C \wedge g)]_R \wedge I(l')\big)^{\nearrow} \wedge I(l')$$

with C' satisfiable.

That is, in the parametric zone graph, nodes are symbolic states, and arcs are labeled by *edges* of the original PTA.

If $\big((l, C), e, (l', C')\big) \in \Rightarrow$, we write $\mathsf{Succ}(\mathbf{s}, e) = (l', C')$.

A graphical illustration of the computation of Succ is given in Figure 2.2.[2]

A symbolic run of a PTA is an alternating sequence of symbolic states and edges starting from the initial symbolic state, of the form $\mathbf{s}_0 \overset{e_0}{\Rightarrow} \mathbf{s}_1 \overset{e_1}{\Rightarrow} \cdots \overset{e_{m-1}}{\Rightarrow} \mathbf{s}_m$, such that for all $i = 0, \ldots, m-1$, $e_i \in E$, $\mathbf{s}_i, \mathbf{s}_{i+1} \in \mathbf{S}$ and $(\mathbf{s}_i, e, \mathbf{s}_{i+1}) \in \Rightarrow$. Given a symbolic state $\mathbf{s}$, we say that $\mathbf{s}$ is reachable if $\mathbf{s}$ belongs to a symbolic run of $\mathcal{A}$. In the following, we simply refer to symbolic states belonging to a run of $\mathcal{A}$ as symbolic states of $\mathcal{A}$.

[2] This figure comes from [AS13], itself coming from an adaptation of a figure by Ulrich Kühne.

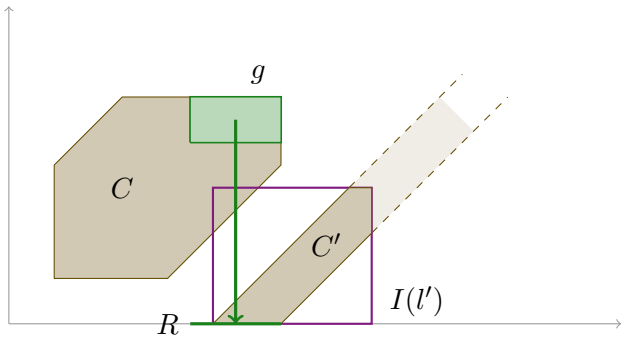


Figure 2.2 – Computing the successor of a symbolic state

Example 2.4. Consider again the coffee machine example in Figure 2.1. A (non-maximal) symbolic run is as follows:
$(\text{idle}, x_1 = x_2 \wedge x_1 \geq 0) \overset{\text{press}}{\mapsto} (\text{addsugar}, x_1 = x_2 \wedge 0 \leq x_2 \leq p_2) \overset{\text{press}}{\mapsto} (\text{sugar}, p_1 \leq x_2 - x_1 \leq p_2 \wedge 0 \leq x_2 \leq p_2) \overset{\text{press}}{\mapsto} (\text{addsugar}, 2 \times p_1 \leq x_2 - x_1 \leq p_2 \wedge 0 \leq x_2 \leq p_2) \overset{\text{cup}}{\mapsto} (\text{preparingcoffee}, 2 \times p_1 \leq x_2 - x_1 \leq p_2 \wedge p_2 \leq x_2 \leq p_3) \overset{\text{coffee}}{\mapsto} (\text{done}, 0 \leq x_1 \leq 10 \wedge x_2 - x_1 = 10 \wedge 2 \times p_1 \leq p_2 \leq p_3) \overset{\text{press}}{\mapsto} (\text{addsugar}, x_1 = x_2 \wedge 0 \leq x_2 \leq p_2 \wedge 2 \times p_1 \leq p_2 \leq p_3)$
The parametric zone graph of this example is infinite.

2.1.3 Subclasses of PTAs

Lower-bound/upper-bound parametric timed automata (L/U-PTAs), proposed in [Hun+02], restrict the use of parameters in the model.

Definition 2.7 (L/U-PTA). An L/U-PTA is a PTA where the set of parameters is partitioned into lower-bound parameters and upper-bound parameters, where an upper-bound (resp. lower-bound) parameter p_i is such that, for every guard or invariant constraint $x \bowtie \sum_{1 \leq j \leq M} \beta_j p_j + d$, we have: $\beta_j > 0$ implies $\bowtie \in \{\leq, <\}$ (resp. $\bowtie \in \{\geq, >\}$), and $\beta_j < 0$ implies $\bowtie \in \{\geq, >\}$ (resp. $\bowtie \in \{\leq, <\}$).

Given an L/U-PTA, we denote by $v_{0/\infty}$ the special parameter valuation (mentioned in, e. g., [Hun+02]) assigning 0 to all lower-bound parameters and ∞ to all upper-bound parameters.[3]

In [BL09], two additional subclasses are introduced: L-PTAs (resp. U-PTAs) are PTAs with only lower-bound (resp. upper-bound) parameters.

L/U-PTAs enjoy a well-known monotonicity property [Hun+02]: increasing upper-bound parameters or decreasing lower-bound parameters can only add behaviors.

Example 2.5. Consider again the coffee machine in Figure 2.1, modeled using a PTA $\mathcal{A}$. This PTA is not an L/U-PTA; indeed, in the guard $x_2 = p_2$ (resp. $x_2 = p_3$), p_2 (resp. p_3) is compared with clocks both as a lower-bound and as an upper-bound. (Recall that $=$ stands for $\leq$ and $\geq$.)
However, if one replaces $x_2 = p_2$ with $x_2 \leq p_2$ and one replaces $x_2 = p_3$ with $x_2 \leq p_3$, then $\mathcal{A}$ becomes

[3] Technically, $v_{0/\infty}$ is not a parameter valuation, as the definition of valuation does not allow ∞. However, we will use it only to valuate an L/U-PTA with it; observe that valuating an L/U-PTA with $v_{0/\infty}$ still gives a valid TA.

an L/U-PTA with lower-bound parameter p_1 and upper-bound parameters $\{p_2, p_3\}$. Note that equalities are not forbidden in L/U-PTAs (e. g., $x_1 = 10$), but only equalities involving parameters.

Several case studies fit into the class of L/U-PTAs: the root contention protocol, the bounded retransmission protocol and the Fischer mutual exclusion protocol are all modeled with L/U-PTAs in [Hun+02]; in [Hun+02; KP12], both the Fischer mutual exclusion protocol and a producer-consumer are verified using L/U-PTAs. Interestingly, the two case studies of the seminal paper on PTAs [AHV93] (viz., a toy train gate controller model and a model of Fischer mutual exclusion protocol) are also L/U-PTAs, although the concept of L/U-PTAs had not yet been proposed at that time. In addition, most models of asynchronous circuits with bi-bounded delays (i. e., where each delay between the change of an input signal and the change of the corresponding output is a parametric interval) can be modeled using L/U-PTAs.

In this manuscript, we will also consider *bounded* PTAs, i. e., PTAs with a bounded parameter domain that assigns to each parameter an infimum and a supremum, both integers.

Definition 2.8 (bounded PTA)**.** A *bounded PTA* is $\mathcal{A}_{|bounds}$, where $\mathcal{A}$ is a PTA, and $bounds$ assigns to each parameter p an interval $[\inf, \sup]$, $(\inf, \sup]$, $[\inf, \sup)$, or $(\inf, \sup)$, with $\inf, \sup \in \mathbb{N}$. We use $\inf(p, bounds)$ and $\sup(p, bounds)$ to denote the infimum and the supremum of p, respectively. (Note that we rule out ∞ as a supremum.)

We say that a bounded PTA is a *closed bounded PTA* if, for each parameter p, its ranging interval $bounds(p)$ is of the form $[\inf, \sup]$; otherwise it is an *open bounded PTA*.

We define similarly bounded L/U-PTAs.

2.2 Decision and computation problems

2.2.1 Decision problems

Acceptance The first problem we will investigate in parametric timed automata is in fact a problem not related to parameters. This problem is called *membership* (in e. g., [Mil00; And18]) but we rename it into *acceptance-problem* so as to avoid confusion with what we call membership in [ALR16a].

acceptance problem:
Input: A PTA $\mathcal{A}$ and a parameter valuation v
Problem: Is $\mathsf{UL}(v(\mathcal{A}))$ empty?

This problem is typically a *timed automata* problem as $v(\mathcal{A})$ is a TA, and can be solved using techniques proposed in [AD94].

Membership The second problem we will investigate in parametric timed automata is the membership problem, that asks whether a PTA belongs to a given subclass of PTAs.

membership problem:
Input: A PTA $\mathcal{A}$ and a subclass of PTAs
Problem: Does $\mathcal{A}$ belong to the given subclass of PTAs?

Deciding the membership of a syntactic subclass of PTAs such as L/U-PTAs is trivial; the membership problem will be harder for some subclasses we will define later though.

Emptiness and universality of the valuations set Now, let us move to fully parametric problems. Let $\mathcal{P}$ be a given a class of decision problems (reachability, unavoidability, etc.).

$\mathcal{P}$-emptiness problem:
INPUT: A PTA $\mathcal{A}$ and an instance ϕ of $\mathcal{P}$
PROBLEM: Is the set of parameter valuations v such that $v(\mathcal{A})$ satisfies ϕ empty?

$\mathcal{P}$-universality problem:
INPUT: A PTA $\mathcal{A}$ and an instance ϕ of $\mathcal{P}$
PROBLEM: For all parameter valuations v, does $v(\mathcal{A})$ satisfy ϕ?

In this manuscript, we mainly focus on the following decision problems:

- reachability (EF[4]): given a TA $v(\mathcal{A})$, is there at least one run of $v(\mathcal{A})$ that reaches a given location? That is, EF-emptiness asks: "is the set of parameter valuations v such that the TA $v(\mathcal{A})$ reaches a given location empty?" And EF-universality asks: "are all parameter valuations such that the corresponding TA reaches a given location?"
- unavoidability (AF): given a TA $v(\mathcal{A})$, do all runs of $v(\mathcal{A})$ eventually reach a given location?
- EG: given a TA $v(\mathcal{A})$ and a subset T of its locations, is there at least one maximal run of $v(\mathcal{A})$ that always stays in T?
- AG: given a TA $v(\mathcal{A})$ and a subset T of its locations, do all runs of $v(\mathcal{A})$ stay in T?
- deadlock-existence (ED): given a TA $v(\mathcal{A})$, is there at least one maximal run of $v(\mathcal{A})$ that is deadlocked, i. e., has no discrete successor (possibly after some delay)?
- cycle-existence (EC): given a TA $v(\mathcal{A})$, is there at least one run of $v(\mathcal{A})$ with an infinite number of discrete transitions?

Note that AF-emptiness is equivalent to EG-universality, while AG-emptiness is equivalent to EF-universality.

We will finally consider the following two additional emptiness problems:

Language-preservation-emptiness problem:
INPUT: A PTA $\mathcal{A}$ and a parameter valuation v'
PROBLEM: Is the set of parameter valuations v such that $v \neq v'$ and for which $v(\mathcal{A})$ has the same untimed language as $v'(\mathcal{A})$ empty?

Trace-preservation-emptiness problem:
INPUT: A PTA $\mathcal{A}$ and a parameter valuation v'
PROBLEM: Is the set of parameter valuations v such that $v \neq v'$ and for which $v(\mathcal{A})$ has the same set of traces as $v'(\mathcal{A})$ empty?

[4]The names "EF", "AF", "EG" come from the TCTL syntax, and are consistent with the notations introduced in [JLR15] and subsequently used in further papers (such as [ALR16a; AL17a]).

2.2.2 Computation problem

Additionally, we define the following computation problem:

$\mathcal{P}$-synthesis problem:
INPUT: A PTA $\mathcal{A}$ and an instance ϕ of $\mathcal{P}$
PROBLEM: Compute the parameter valuations such that $v(\mathcal{A})$ satisfies ϕ.

Example 2.6. Let us exemplify some decision and computation problems for the PTA in Figure 2.1. Assume the unique target location is "done", i. e., $T = \{\text{done}\}$. EF-emptiness asks whether the set of parameter valuations that can reach location "done" for some run is empty; this is false (e. g., $p_1 = 1$, $p_2 = 2$, $p_3 = 3$ can reach "done"). EF-universality asks whether all parameter valuations can reach location "done" for some run; this is false (no parameter valuation such that $p_2 > p_3$ can reach "done"). AF-emptiness asks whether the set of parameter valuations that can reach location "done" for all runs is empty; this is false (e. g., $p_1 = 1$, $p_2 = 2$, $p_3 = 3$ cannot avoid "done"). EF-synthesis consists in synthesizing all valuations for which a run reaches location "done"; the resulting set of valuations is $0 \leq p_2 \leq p_3 \leq 10 \wedge p_1 \geq 0$.

2.3 A semi-algorithm for reachability synthesis

I recall in Algorithm 1 the semi-algorithm EFsynth.[5] If it terminates, EFsynth synthesizes all parameter valuations for which a set of location T is reachable, and therefore it is a correct solution to the EF-synthesis problem. This semi-algorithm was proved correct (sound and complete) in [JLR15]. Although several procedures to solve EF-synthesis were proposed in the literature (for example [AHV93; Hun+02]), we choose to give this algorithm for several reasons:

1. It manipulates the same semantics as in Definition 2.6;
2. A full proof of correctness is available in [JLR15];
3. We will modify it subsequently in Section 4.3.

Algorithm 1: $\mathsf{EFsynth}(\mathcal{A}, s, T, S)$

input : A PTA $\mathcal{A}$, a symbolic state $s = (l, C)$, a set of target locations T, a set S of passed states on the current path
output: Constraint K over the parameters

```
if l ∈ T then K ← C↓_P ;
else
    K ← ⊥;
    if s ∉ S then
        for each outgoing edge e from l in A do
            K ← K ∪ EFsynth(A, Succ(s, e), T, S ∪ {s});
return K
```

[5] Recall that a semi-algorithm is a procedure that may not terminate but, if it does, then its result is exact, i. e., sound and complete.

EFsynth proceeds as a post-order traversal of the symbolic reachability tree, and collects all parametric constraints associated with the target locations T. The variable S serves to remember the visited states. The initial call to EFsynth is $\mathsf{EFsynth}(\mathcal{A}, s_0, T, \emptyset)$.

If the state is a target sate, the projection of its constraint onto the parameters is returned (line 1). Otherwise, the algorithm returns the union over all outgoing edges of the algorithm recursively applied to its successors via these edges (line 6).

Proposition 2.1 ([JLR15]). *Assume EFsynth$(\mathcal{A}, s_0, T, \emptyset)$ terminates with result K. Then $v \models K$ iff T is reachable in $v(\mathcal{A})$.*

2.4 A semi-algorithm for trace set preservation synthesis

I recall in Algorithm 2 the *inverse method*, originally proposed in a non-deterministic (and potentially incomplete) version in [And+09b], and extended to a complete result in [AM15]. This is mainly a work originating from my PhD thesis, and therefore not a contribution of this habilitation manuscript. However, some of the works described in the following chapters partially rely on IM (namely Sections 4.4 and 5.1). The goal of IM is, given a PTA $\mathcal{A}$ and a parameter valuation v, to synthesize all other valuations giving the same trace set as $v(\mathcal{A})$.

IM relies on the following notion of v-compatibility.

Definition 2.9 (v-compatibility). A symbolic state $\mathbf{s} = (l, C)$ of a PTA is v-compatible if $v \models C$.

Algorithm 2: $\mathsf{IM}(\mathcal{A}, v)$

input : A PTA $\mathcal{A}$, a parameter valuation v
output: Constraint K over the parameters

1 $K_{good} \leftarrow \top$; $K_{bad} \leftarrow \bot$; $\mathbf{S}_{new} \leftarrow \{s_0\}$; $\mathbf{S} \leftarrow \varnothing$
2 **while** *true* **do**
3 **foreach** *state* $(l, C) \in \mathbf{S}_{new}$ **do**
4 **if** $v \models C{\downarrow_P}$ **then** $K_{good} \leftarrow K_{good} \wedge C{\downarrow_P}$;
5 **else** $K_{bad} \leftarrow K_{bad} \vee C{\downarrow_P}$; $\mathbf{S}_{new} \leftarrow \mathbf{S}_{new} \setminus \{(l, C)\}$;
6 **if** $\mathbf{S}_{new} \subseteq \mathbf{S}$ **then return** $K_{good} \wedge \neg K_{bad}$;
7 $\mathbf{S} \leftarrow \mathbf{S} \cup \mathbf{S}_{new}$; $\mathbf{S}_{new} \leftarrow \mathsf{Succ}(\mathbf{S}_{new})$

IM maintains two constraints: K_{good} is the intersection of the parameter constraints associated with the v-compatible states met, whereas K_{bad} is the union[6] of the parameter constraints associated with all v-incompatible states. IM also maintains two sets of states, viz., the set $\mathbf{S}$ of all states met, and the set $\mathbf{S}_{new}$ of states met at the latest iteration of the **while** loop. IM is a breadth-first search algorithm exploring the parametric zone graph of $\mathcal{A}$. Whenever a new state is met, its v-compatibility is checked (line 4). If it is v-compatible, its projection onto the parameters is added to K_{good} (line 4). Otherwise, its projection onto the parameters is added to K_{bad} (line 5), and the state is discarded from $\mathbf{S}_{new}$ (line 5), i. e., its successors will not be explored. When no new states can be explored, i. e., the set $\mathbf{S}_{new}$ is either empty or contains only

[6] This union of constraints can be seen (and implemented) as a finite list of convex constraints.

states explored earlier (line 6), the intersection of v-compatible parametric constraints and the negation of the v-incompatible parametric constraints is returned (line 6). Otherwise, the algorithm explores one step further in depth (line 7). As an abuse of notation, we use $\mathsf{Succ}(\mathbf{S}_{new})$ to denote the set of successors of all states (l, C) in $\mathbf{S}_{new}$, for all edges outgoing from l.

Proposition 2.2 ([And+09b; AM15]). *Let $\mathcal{A}$ be a deterministic PTA, and let v be a parameter valuation. Assume IM$(\mathcal{A}, v)$ terminates with result K. Then $v' \models K$ iff $\mathsf{Traces}(v(\mathcal{A})) = \mathsf{Traces}(v'(\mathcal{A}))$.*

Chapter 3

Decidability and expressiveness of parametric timed automata

Parametric timed automata are a very powerful formalism with a high expressiveness, which comes with the drawback that most interesting problems are undecidable. Studying decidability may not be a goal in its own end, but is useful to determine whether some problems can be solved. When a problem is undecidable, it is hopeless to look for exact algorithms—although designing semi-algorithms or procedure returning an approximate result can still be handful (this will be the subject of Chapter 5). We first survey the existing decidability results for parametric timed automata (Section 3.1); then, we propose a new definition of the expressiveness of parametric timed automata (Section 3.2); finally, we significantly enhance the knowledge we have on parametric timed automata by studying several problems, and exhibiting new decidable subclasses (Section 3.3).

3.1 Exploring the jungle of decidability results

Since [AHV93], numerous (un)decidability results have been proposed, in various settings. Our first contribution is to summarize the state-of-the-art knowledge of PTAs for several problems. (A complete version of this survey is available in [And18].)

3.1.1 Almost everything is undecidable for simple PTAs

Simple PTAs We use the following class of *simple PTAs* as the subclass of PTAs where guards and invariants are simple constraints. We propose this class to recall that, even in this restricted situation, all non-trivial problems are undecidable.

In this entire subsection, we consider simple PTAs without restriction on the number of clocks and parameters. In that situation, all non-trivial problems studied in the literature are undecidable, with the exception of the acceptance problem—which is rather a problem for TAs. By non-trivial, we mean requiring a semantic analysis, and not, e. g., a sole analysis of the syntax of the PTA (e. g., "is the number of clocks even", or any problem defined in Section 2.2 by setting $T = L$ or $T = \emptyset$).

Whereas bounding time or bounding the parameter domain for rational-valued parameters preserves the undecidability, we will recall in Section 3.1.2 that bounding the number of clocks and/or parameters brings decidability.

All proofs of undecidability reduce from either the halting problem, or the boundedness problem, of a 2-counter machine, both known to be undecidable [Min67].

Decidability of the acceptance

The acceptance problem is not strictly speaking a problem for PTAs, but rather for TAs, since it considers a valuated PTA.

On the one hand, the acceptance problem is decidable (and PSPACE-complete) for PTAs over discrete time, over dense time with integer-valued parameters, and over dense time with rational-valued parameters [AD94].

On the other hand, the acceptance problem becomes undecidable with real-valued (in fact irrational) parameters. Indeed, the reachability of a location in a TA with irrational constants is undecidable [Mil00]. The idea is to encode a 2-counter machine using 2 clocks x_1 and x_2 (plus an additional third clock), where the value c_i of counter i is encoded using $x_i = c_i \times \tau$, for $i \in \{1, 2\}$, with τ the irrational constant (the value $\sqrt{2}$ is suggested for τ).

General undecidable problems

EF-emptiness The seminal paper on PTAs [AHV93] showed that the EF-emptiness problem is undecidable for PTAs, both for discrete time and for dense-time.

AF-emptiness In [JLR15], it is proved that the AF-emptiness is undecidable for L/U-PTAs with 3 clocks and 4 integer-valued parameters, and hence for PTAs as well.

Bounding time

Bounded-time model checking consists in checking a property *within a bounded time domain*. Undecidable problems might become decidable in this situation, or be of a lower complexity. For example, the language inclusion for timed automata becomes decidable over bounded-time [OW10], although it is undecidable in general. In addition, time-bounded reachability becomes decidable for a special subclass of hybrid automata with monotonic (either non-negative or non-positive) rates [Bri+13], although it is undecidable in general.

In contrast, the EF-emptiness problem remains undecidable for (general) PTAs over bounded, dense time [Jov13, Theorem 3.4].

Bounding the parameter domain

Decidability for integer-valued parameters The $\mathcal{P}$-emptiness problem for PTAs with bounded integers is PSPACE-complete for any class of problems $\mathcal{P}$ that is PSPACE-complete for TAs [JLR15]. As a consequence, EF-, AF-, EG-, AG-emptiness are all decidable; and so are language and trace preservation. More generally, the whole TCTL model checking, including reachability and unavoidability, is PSPACE-complete [ACD93], and therefore the corresponding emptiness problems are PSPACE-complete for PTAs with bounded integer parameters.

In [JLR15], a symbolic method is proposed to compute EF- and AF-synthesis; experiments showed that this symbolic computation is faster than an exhaustive enumeration (using Uppaal).

Undecidability for rational-valued parameters For rational-valued parameters, the EF-emptiness problem is undecidable for a single parameter in $[1, 2]$ [Mil00].

$\mathbb{T}$	$\mathbb{P}$	Guards	Invariants	P-clocks	NP-clocks	Params	Decidability	Main ref.
$\mathbb{N}$	$\mathbb{N}$	$x \bowtie p\|d$		1	0	fixed	(at most) PTIME	[Mil00] (consequence)
$\mathbb{N}$	$\mathbb{N}$	$x \bowtie p\|d$		1	0	any	(at most) NP-complete	[Mil00] (consequence)
$\mathbb{N}$	$\mathbb{N}$	$x \leq\geq p\|d^+$		1	any	any	NEXPTIME-complete	[BO14]
$\mathbb{N}$	$\mathbb{N}$	$x \bowtie p\|d$	$x \preceq p\|d^+$	1	any	any	(at most) NEXPTIME	[Ben+15] (consequence)
$\mathbb{N}$	$\mathbb{N}$	$x \leq\geq p\|d^+$		2	any	1	PSPACE$^{\text{NEXP}}$-hard	[BO14]
$\mathbb{N}$	$\mathbb{N}$	any		2	any	> 1	open	
$\mathbb{N}$	$\mathbb{N}$	$x \bowtie p\|d$	None	3	0	1	undecidable	[Ben+15]
$\mathbb{N}$	$\mathbb{N}$	$x = p\|d$	None	3	0	6	undecidable	[AHV93]
$\mathbb{N}$	$\mathbb{N}$	$x <> p$		any	any	any	open	
$\mathbb{N}$	$\mathbb{N}$ bounded	$x \bowtie plt$	$x \preceq plt$	any	any	any	(at most) PSPACE-complete	[JLR15] (consequence)
$\mathbb{R}_+$	$\mathbb{N}$	$x \bowtie p\|d$		1	0	fixed	(at most) PTIME	[Mil00] (consequence)
$\mathbb{R}_+$	$\mathbb{N}$	$x \bowtie p\|d$		1	0	any	(at most) NP-complete	[Mil00] (consequence)
$\mathbb{R}_+$	$\mathbb{N}$	$x \bowtie p\|d$	$x \preceq p\|d^+$	1	any	any	NEXPTIME	[Ben+15]
$\mathbb{R}_+$	$\mathbb{N}$	any		2	any	any	open	
$\mathbb{R}_+$	$\mathbb{N}$	$x \bowtie p\|d$	None	3	0	1	undecidable	[Ben+15]
$\mathbb{R}_+$	$\mathbb{N}$	$x = p\|d$	None	3	0	6	undecidable	[AHV93] (consequence)
$\mathbb{R}_+$	$\mathbb{N}$	$x <> p$		any	any	any	open	
$\mathbb{R}_+$	$\mathbb{N}$ bounded	$x \bowtie plt$	$x \preceq plt$	any	any	any	PSPACE-complete	[JLR15]
$\mathbb{R}_+$	$\mathbb{Q}_+$	$x \bowtie p\|d$		1	0	fixed	PTIME	[Mil00]
$\mathbb{R}_+$	$\mathbb{Q}_+$	$x \bowtie p\|d$		1	0	any	NP-complete	[Mil00]
$\mathbb{R}_+$	$\mathbb{Q}_+$	any		1	1 or 2	any	open	
$\mathbb{R}_+$	$\mathbb{Q}_+[1;2]$	$x \bowtie p\|d$		1	3	1	undecidable	[Mil00]
$\mathbb{R}_+$	$\mathbb{Q}_+$	any		2	0 or 1	any	open	
$\mathbb{R}_+$	$\mathbb{Q}_+[1;2]$	$x \bowtie p\|d$		2	2	1	undecidable	[Mil00] (consequence)
$\mathbb{R}_+$	$\mathbb{Q}_+[1;2]$	$x \bowtie p\|d$		3	0	1	undecidable	[Mil00]
$\mathbb{R}_+$	$\mathbb{R}_+$	$x = p\|d$	None	3	0	6	undecidable	[AHV93]
$\mathbb{R}_+$	$\mathbb{Q}_+$	$x <> p$		< 2	3	2	open	
$\mathbb{R}_+$	$\mathbb{Q}_+$	$x <> p$		2	< 3	2	open	
$\mathbb{R}_+$	$\mathbb{Q}_+$	$x <> p$		2	3	< 2	open	
$\mathbb{Q}_+/\mathbb{R}_+$	$\mathbb{Q}_+/\mathbb{R}_+$	$x <> p$		2	3	2	undecidable	[Doy07]

Table 3.1 – Decidability of the EF-emptiness problem for general PTAs

3.1.2 Bounding the numbers of clocks and parameters

EF-emptiness

Since [AHV93], the decidability of the EF-emptiness problem was studied in various settings, by bounding the number of parametric clocks, of non-parametric clocks, and of parameters. The syntax was also restrained. We summarize these results in Table 3.1 ($\mathbb{T}$ and $\mathbb{P}$ denote the domain of time and of parameter valuations respectively).

We need to consider not only the number of clocks and parameters, but also the syntax allowed in guards and invariants. For example, for the undecidability over discrete time, [Ben+15] improves the number of parameters when compared to [AHV93] (6 instead of 1), but requires both strict and non-strict inequalities, whereas [AHV93] uses only equalities in their construction; it is therefore unclear whether the result of [AHV93] is really subsumed by [Ben+15].

"Consequence" indicates a result originally proved for a less expressive or a more expressive setting; "at most" in the complexity column indicates in the latter case that the complexity is necessarily lower or equal to that of the more expressive setting. For example, [Mil00] proved that the single clock case is PTIME over dense time with a fixed number of rational-valued parameters, and therefore the corresponding problem cannot be harder over discrete time.

Let us now extract the most important results out of Table 3.1. The decidability is clearly impacted by

the number of parametric clocks, and we therefore reason by the number of parametric clocks.

Main results: 1 parametric clock First, let us consider PTAs with a single parametric clock: The EF-emptiness problem is (at most) NP-complete over discrete and dense time with no non-parametric clock and arbitrarily many parameters [Mil00].

It is decidable for over discrete time with arbitrarily many non-parametric clocks (NEXPTIME-complete when only non-strict inequalities are used [BO14], and at most NEXPTIME when both strict and non-strict inequalities are used but with invariants of the form $x \preceq p|d^+$ [Ben+15]). Over dense-time with arbitrarily many non-parametric clocks and integer-valued parameters, it is NEXPTIME.

It is undecidable with three non-parametric clocks [Mil00] over dense time with rational-valued parameters; note that this problem is decidable over discrete time [AHV93; BO14; Ben+15] and over dense time with integer-valued parameters [Ben+15], which exhibits a difference between dense and discrete time [Mil00], as well as between integer- and rational-valued parameters over dense time.

Main results: 2 parametric clocks The EF-emptiness problem is $\text{PSPACE}^{\text{NEXP}}$-hard [BO14] over discrete time with two parametric clocks and a single parameter. Over dense-time with rational-valued parameters, the case with 2 parametric clocks and 2 non-parametric clocks is undecidable. Any other case with two parametric clocks remains open.

Main results: other undecidability The EF-emptiness problem is undecidable in all settings with three (or more) parametric clocks.

Finally, using only strict inequalities, the EF-emptiness problem is undecidable over dense time for two parametric clocks, three non-parametric clocks and two parameters [Doy07]; this situation was not considered over discrete time.

Open cases The main open case is the "two parametric clocks" case. The decidability is open for 2 parametric clocks with:

- over discrete-time: arbitrarily many non-parametric clocks and more than one parameter;
- over dense-time with integer-valued parameters: arbitrarily many non-parametric clocks and parameters;
- over dense-time with rational-valued parameters: 0 or 1 non-parametric clock and arbitrarily many parameters.

In addition, the decidability remains open over dense-time with rational-valued parameters for 1 non-parametric clock, 1 or 2 non-parametric clocks and arbitrarily many parameters.

Finally, the decidability using only strict inequalities remain open for cases not considered by [Doy07]: less clocks and parameters, or with integer-valued parameters.

3.1.3 L/U-PTAs

A main decidability result

The first (and main) positive result for L/U-PTAs is the decidability of the EF-emptiness problem [Hun+02]. L/U-PTAs benefit from the following interesting monotonicity property: increasing the value of an upper-bound parameter or decreasing the value of a lower-bound parameter necessarily relaxes the guards and invariants, and hence can only add behaviors. Hence, checking the EF-emptiness of an L/U-PTA can be

achieved by replacing all lower-bound parameters with 0, and all upper-bound parameters with a sufficiently large constant; this yields a non-parametric TA, for which emptiness is PSPACE-complete [AD94].

Further decidability results are exhibited in [BL09], for infinite runs acceptance properties, i. e., where a location is visited infinitely often. Note that, in contrast to [Hun+02] where the parameters are valued with non-negative reals, the results in [BL09] consider integer-valued parameters (though time is dense, i. e., clocks are real-valued). It is shown in [BL09] that emptiness, universality, finiteness of the valuation set are PSPACE-complete for infinite runs acceptance properties.

Undecidability results

The first undecidability results for L/U-PTAs are shown in [BL09]: the *constrained* EF-emptiness problem and constrained EF-universality problem (for infinite runs acceptance properties) are undecidable for L/U-PTAs. By constrained it is meant that some parameters of the L/U-PTA can be constrained by an initial linear constraint, e. g., $p_1 \leq 2 \times p_2 + p_3$. Indeed, using linear constraints, one can constrain an upper-bound parameter to be equal to a lower-bound parameter, and hence build a 2-counter machine using an L/U-PTA. However, when no upper-bound parameter is compared to a lower-bound parameter (i. e., when no initial linear inequality contains both an upper-bound and a lower-bound parameter), these two problems retrieve decidability [BL09].

A second negative result is shown in [JLR15]: the AF-emptiness problem is undecidable for L/U-PTAs. This is achieved by a reduction from a 2-counter machine where a lower-bound parameter is equal to an upper-bound parameter iff AF holds. This restricts again the use of L/U-PTAs, as AF is essential to show that all possible runs of a system eventually reach a (good) state.

Intractability of the synthesis

The most disappointing result concerning L/U-PTAs is shown in [JLR15]: despite decidability of the underlying decision problems (EF-emptiness and EF-universality), the solution to the EF-synthesis problem for L/U-PTAs, if it can be computed, cannot be represented using a formalism for which the emptiness of the intersection with equality constraints is decidable. A very annoying consequence is that such a solution cannot be represented as a finite union of polyhedra (since the emptiness of the intersection with equality constraints is decidable).

3.2 Expressiveness of parametric timed automata

After surveying decidability in the previous section, we now compare subclasses of parametric timed automata with each other. This implies to define a notion of expressiveness, which was not done before. We will in fact propose two different definitions. But before that, let us introduce a new subclass of PTAs.

3.2.1 A new subclass: Integer-point parametric timed automata

Definition 3.1. A PTA $\mathcal{A}$ is an *integer points PTA* (in short *IP-PTA*) if, in any reachable symbolic state (l, C) of $\mathcal{A}$, C contains at least one integer point, i. e., $\exists v : P \to \mathbb{N}, \exists w : X \to \mathbb{N}$ s.t. $w|v \models C$.

Example 3.1. Consider the PTA in Figure 3.1a. This PTA is not an IP-PTA; indeed, the (unique) symbolic state with location l_3 contains only $\frac{1}{2}$ in dimension p, and this symbolic state therefore contains no integer point.

In contrast, it can be shown that the PTA in Figure 3.1b is an IP-PTA. The coffee machine in Figure 2.1 (which has an infinite parametric zone graph) is also an IP-PTA.

Comparison with L/U-PTAs

Proposition 3.1 ([ALR16a]). *The class of IP-PTAs is incomparable with the class of L/U-PTAs.*

Proof idea.
- Consider an L/U-PTA with a transition guarded by $x > 0$ and resetting no clock, followed by a second location with invariant $x < 1$; then, necessarily, the symbolic state associated with this second location contains no integer point (as $x \in (0, 1)$ in that symbolic state).
- It is easy to exhibit an IP-PTA that is not an L/U-PTA. This is for example the case of Figures 2.1 and 3.1b.

□

However, we can prove that any *non-strict* L/U-PTA, i. e., with only non-strict inequalities, is an IP-PTA. This gives that the class of non-strict L/U-PTAs is included in IP-PTAs. With additional results, we get the following comparison:

Theorem 3.1 ([ALR16b]). *The class of IP-PTAs is strictly larger than the class of non-strict L/U-PTAs. The class of bounded IP-PTAs is strictly larger than the class of non-strict bounded L/U-PTAs. The class of bounded IP-PTAs is incomparable with the class of bounded L/U-PTAs. The class of bounded IP-PTAs is incomparable with the class of L/U-PTAs.*

Membership

A main disappointing result is that the membership problem is undecidable for IP-PTAs, even when bounded. In other words: it is not possible to decide in general whether a PTA is an IP-PTA.

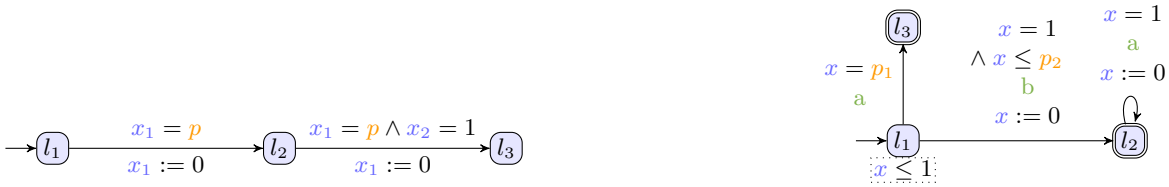
Theorem 3.2 ([ALR16a]). *It is undecidable whether a PTA is an IP-PTA, even when bounded.*

Proof idea. The proof reduces from the halting problem of a 2-counter machine: we consider an existing encoding [ALR16a], show that there is an integer point in any symbolic state, and make a slight modification as follows: from the location encoding the halting state of the 2-counter machine, we add a transition to a state that contains no integer point (which can easily be enforced using a constraint such as $0 < x < 1$ for some clock x). As a consequence, this PTA is an IP-PTA iff the 2-counter machine does not halt. □

Due to the undecidability of membership, the class of IP-PTAs may not seem very interesting. However, this subclass will help us to exhibit further undecidability for other subclasses of PTAs in Section 3.3.

Theorem 3.1 provides a sufficient syntactic membership condition. In addition, we now define another new non-trivial set of restrictions leading to IP-PTAs.

Definition 3.2 (Reset-PTA [ALR16a]). A *reset-PTA* is a PTA where, for each transition, if at least one parameter appears in the transition guard or in the source location invariant, then all clocks are reset along this transition.



(a) A PTA which is not an IP-PTA

(b) A PTA which is an IP-PTA

Figure 3.1 – Examples of PTA

This kind of restriction is somewhat reminiscent of those enforced by *initialized* hybrid automata [Hen+98] to obtain decidability. We now prove that reset-PTAs are IP-PTAs, which in turn means that any decidable problem for IP-PTAs (which will be studied in Section 3.3) is also decidable for reset-PTAs.

Theorem 3.3 ([ALR16a]). *Any reset-PTA is an IP-PTA.*

3.2.2 Defining the expressiveness of parametric timed automata

No paper has compared the expressiveness of PTAs and their subclasses, nor even proposed a definition of the expressiveness. We propose here two definitions.

In the following, we denote by $\mathcal{V}(P)$ the set of valuations of all the parameters in P.

Definition 3.3 (untimed language of a PTA)**.** The *untimed language* of a PTA $\mathcal{A}$, denoted by $\mathsf{UL}(\mathcal{A})$ is the union over all parameter valuations v of the sets of untimed words accepted by $v(\mathcal{A})$, i. e.,

$$\bigcup_{v \in \mathcal{V}(P)} \Big\{ \eta \mid \eta \in \mathsf{UL}\big(v(\mathcal{A})\big) \Big\}$$

We also propose a second definition of language, in which we consider not only the accepting untimed words, but also the parameter valuations associated with these words; this definition is more suited to compare the possibilities offered by parameter synthesis.

Definition 3.4 (constrained untimed language of a PTA)**.** The *constrained untimed language* of a PTA $\mathcal{A}$, denoted by $\mathsf{CUL}(\mathcal{A})$, is

$$\bigcup_{v \in \mathcal{V}(P)} \Big\{ (\eta, v) \mid \eta \in \mathsf{UL}\big(v(\mathcal{A})\big) \Big\}$$

We use the word "constrained" because another way to represent the constrained language of a PTA is in the form of a set of elements (η, K), where η is an untimed word, and K is a parametric constraint such that for all v in K, then η is an untimed word accepted by $v(\mathcal{A})$.

Example 3.2. Let us consider the PTA $\mathcal{A}$ of Figure 3.1b.

- Its untimed language is $\mathsf{UL}(\mathcal{A}) = \{a\} \cup \{ba^n \mid n \in \mathbb{N}\}$ that we note with the rational expression

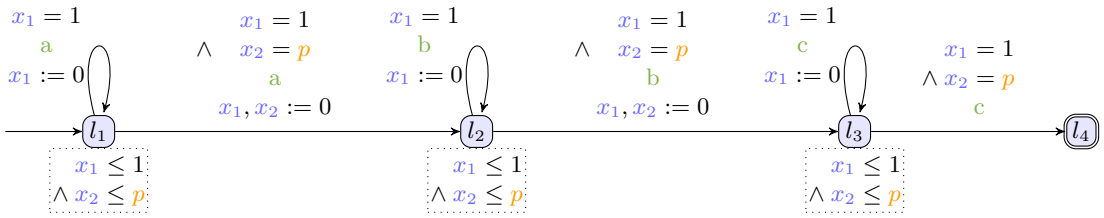


Figure 3.2 – A PTA with untimed language $a^n b^n c^n$

$\mathsf{UL}(\mathcal{A}) = a + ba^*$.

- Its constrained untimed language is $\mathsf{CUL}(\mathcal{A}) = \Big\{(a, p_1 \leq 1), (ba^*, p_1 \geq 0 \wedge p_2 \geq 1)\Big\}$.

Note that the idea of combining the untimed language with the parameter valuations leading to it is close to the idea of the behavioral cartography of parametric timed automata [AF10], that consists in computing parameter constraints together with a trace set.

In the following, a *class* refers to an element in the set of TAs, bounded L/U-PTAs, L/U-PTAs, bounded PTAs and PTAs. An *instance* of a class is a model of that class.

3.2.3 Comparison of the expressiveness of subclasses of PTAs

Let us now compare the expressiveness of various classes w.r.t. the two definitions we proposed above. **We compare here only integer-valued parameters.** Some of the results below may not hold for rational-valued parameters—this is the subject of future work.

Expressiveness as the union of untimed languages

PTAs in the Hierarchy of Chomsky The following result states that Turing-recognizable languages (type-0 in Chomsky's hierarchy) can be recognized by PTAs (with enough clocks and parameters), and derives from the fact that several 2-counter machine encodings using PTAs were proposed in the literature.

Lemma 3.1 ([ALR16b]). *Turing-recognizable languages are also recognizable by PTAs.*

Lemma 3.1 only holds with enough clocks and parameters, typically 3 parametric clocks and 1 integer-valued or rational-valued parameter [Ben+15], or 1 parametric clock, 3 non-parametric clocks and 1 rational-valued parameter [Mil00] (see Section 3.1).

In [AM15, Theorem 20], we proved that the parametric zone graph of a PTA with a single (parametric) clock and arbitrarily many parameters is finite. This gives that the language recognized by a PTA with a single clock is regular.

Lemma 3.2 ([ALR16b]). *The untimed language recognized by a PTA with a single clock and arbitrarily many parameters is regular.*

We now show that adding to the setting of Lemma 3.2 a single non-parametric clock, even with a single parameter, may give a language that is at least context-sensitive, hence beyond the class of regular languages. Consider the PTA $\mathcal{A}$ in Figure 3.2. Consider an integer parameter valuation v such that $v(p) = i$, with $i \in \mathbb{N}$. The idea is that we use the parameter to first count the number of as, and then ensure that

we perform an identical number of bs and cs; such counting feature is not possible in TAs (at least not for any value of i as is the case here). Clearly, due to the invariant $x_1 \leq 1$ in l_1, one must take the self-loop on l_1 every 1 time unit; then, one can take the transition to l_2 only after i such loops. The same reasoning applies to locations l_2 and l_3. Hence, the language accepted by the TA $v(\mathcal{A})$ is $a^{i+1}b^{i+1}c^{i+1}$.

Hence the union over all parameter valuations of the words accepted by $\mathcal{A}$ is $\{a^n b^n c^n \mid n \geq 1\}$. This language is known to be in the class of context-sensitive languages (type-1 in Chomsky's hierarchy), hence beyond the class of regular languages (type-3).

Theorem 3.4 ([ALR16b]). *PTAs with 1 parametric clock, 1 non-parametric clock and 1 parameter can recognize languages that are context-sensitive.*

This result is interesting for several reasons. First, it shows that adding a single clock, even non-parametric, to a PTA with a single clock immediately increases its expressiveness. Second, it falls into the interesting class of PTAs with 2 clocks, for which many problems remain open: the PTA exhibited in the proof of Theorem 3.4 (1 parametric clock and 1 non-parametric) falls into the class of 1 parametric clock, arbitrarily many non-parametric clocks and arbitrarily many integer-valued parameters, for which the EF-emptiness is known to be decidable [Ben+15]. When replacing the integer-valued with a rational-valued parameter (which does not fundamentally change our example), it also falls into the class of 1 parametric clock, 1 non-parametric clock and 1 rational-valued parameter, for which the EF-emptiness is known to be open (see Table 3.1). In both cases, it gives a lower bound on the class of languages recognized by such a PTA.

Expressiveness of PTAs and subclasses First, we can show using the monotonicity property of L/U-PTAs that the untimed language of an L/U-PTA $\mathcal{A}$ is equal to that of the same L/U-PTA valuated with $v_{0/\infty}$.

Lemma 3.3 ([ALR16b]). *Let $\mathcal{A}$ be an L/U-PTA. Then:* $\mathsf{UL}(\mathcal{A}) = \mathsf{UL}(v_{0/\infty}(\mathcal{A}))$.

In addition, since the untimed words recognized by TA form a regular language [AD94], then the PTA exhibited in Theorem 3.4 recognizes a language not recognized by any TA. Conversely, any TA is a PTA (with no parameter) which gives that the expressiveness of PTAs is strictly larger than that of TAs.

Also note that, as we consider integer-valued parameters, there is a finite number of valuations in a bounded PTA. Since the language recognized by a TA is a regular language, and the class of regular languages is closed under finite union, then bounded PTAs also recognize regular languages, and are therefore equally expressive with TAs.

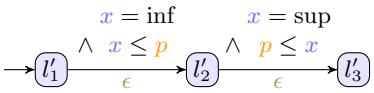
Finally, the 2-counter machine of [AHV93] is an IP-PTA (see [ALR16a] for a detailed argument), which gives that Turing-recognizable languages are also recognizable by IP-PTAs.

This all together gives the following result:

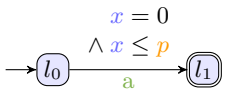
Proposition 3.2 ([ALR16b]). *TAs, L/U-PTAs and bounded PTAs are equally expressive w.r.t. the union of untimed languages.*

PTAs and IP-PTAs are equally expressive, and are strictly more expressive than TAs w.r.t. the union of untimed languages.

We summarize the results in Figure 3.4a.



(a) Bounding a PTA



(b) PTA accepting a for any valuation

Figure 3.3 – A PTA gadget and a PTA

Expressiveness as constrained untimed language

Let us now compare subclasses of PTAs w.r.t. the constrained untimed language.

Bounded PTAs can easily be simulated using a non-bounded PTA, by bounding the parameters using one clock and appropriate extra locations and transitions prior to the original initial location of the PTA. For example, if x is reset when entering l'_1, the gadget in Figure 3.3a ensures that $p \in [\inf, \sup]$. All such gadgets (one per parameter) must be added in a sequential manner, resetting x prior to each gadget, and resetting all clocks when entering the original initial location after the last gadget.[1]

Now, it is easy to find a PTA that has a larger constrained untimed language than any bounded PTA. This is the case of any PTA for which a word is accepting for parameter valuations arbitrarily large (e. g., Figure 3.3b).

This gives the following result:

Proposition 3.3 ([ALR16b]). *Bounded PTAs are strictly less expressive than PTAs w.r.t. the constrained untimed language.*

We now show that, interestingly, this result does not extend to L/U-PTAs, i. e., bounded L/U-PTAs are not strictly less expressive than but incomparable with L/U-PTAs. On the one hand, let us show that the constrained untimed language of a given bounded L/U-PTA cannot be obtained for any L/U-PTA. Consider a bounded U-PTA with a single parameter p^+ with bounds such that $p^+ \in [0, 1]$, and accepting a for any valuation of $p^+ \in [0, 1]$. From the monotonicity of L/U-PTAs, if this run is accepted in an L/U-PTA $\mathcal{A}'$, then this run is also accepted for any valuation v' such that $v'(p^+) \geq 0$, including for instance $v'(p^+) > 1$. Hence accepting a only for valuations of $p^+ \in [0, 1]$ cannot be obtained in an L/U-PTA, and therefore no L/U-PTA yields this constrained untimed language.

This converse is immediate: assume an L/U-PTA with a single parameter p^+, accepting a for any valuation of $p^+ \in [0, \infty)$. From the definition of bounded (L/U-)PTAs, all parameters must be bounded, and therefore there exists no bounded L/U-PTA that can accept a run for $p^+ \in [0, \infty)$. Hence no bounded L/U-PTA yields this constrained untimed language.

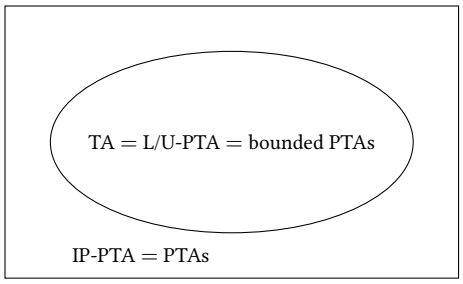
This gives the following result:

Proposition 3.4 ([ALR16b]). *Bounded L/U-PTAs are incomparable with L/U-PTAs w.r.t. the constrained untimed language.*

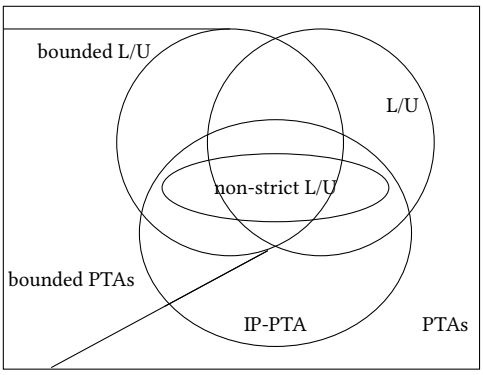
A consequence is that undecidability results for bounded L/U-PTAs cannot be automatically extended to L/U-PTAs; conversely, decidability results for L/U-PTAs cannot be automatically extended to bounded L/U-PTAs.

We summarize the results in Figure 3.4b.

[1] Technically, this does *not* preserve the timed language due to the time needed to pass the initial gadgets; but observe that the untimed language is preserved, provided silent transitions are allowed.



(a) w.r.t. the untimed language



(b) w.r.t. the constrained untimed language

Figure 3.4 – Expressiveness of PTAs and subclasses

What's beyond...? In [ALR16b], we also considered the case of *hidden* parameters, that do not appear in the constrained untimed language (i. e., they are hidden by existential quantification). Hidden parameters do not increase the expressiveness of L/U-PTAs. However, PTAs extended with hidden parameters strictly increase the expressiveness of PTAs.

In addition, we also showed in [ALR16b] that neither PTAs, nor PTAs allowing parametric linear terms (i. e., $plt \bowtie 0$ or $plt \bowtie x$) in guards and invariants, augment the expressiveness w.r.t. the constrained untimed language of PTAs with the most restrictive syntax ($x \bowtie p$) but extended with hidden parameters.

3.3 Decidability of parametric timed automata

After investigating the state-of-the-art results concerning decision problems for parametric timed automata and subclasses in Section 3.1, we now propose a set of results that considerably increase the knowledge we have of this formalism. While (unsurprisingly) all the decision problems are undecidable for general parametric timed automata, we exhibit several subclasses for which they turn decidable, with sometimes subtle differences between the decidable and the undecidable class.

3.3.1 EF-emptiness

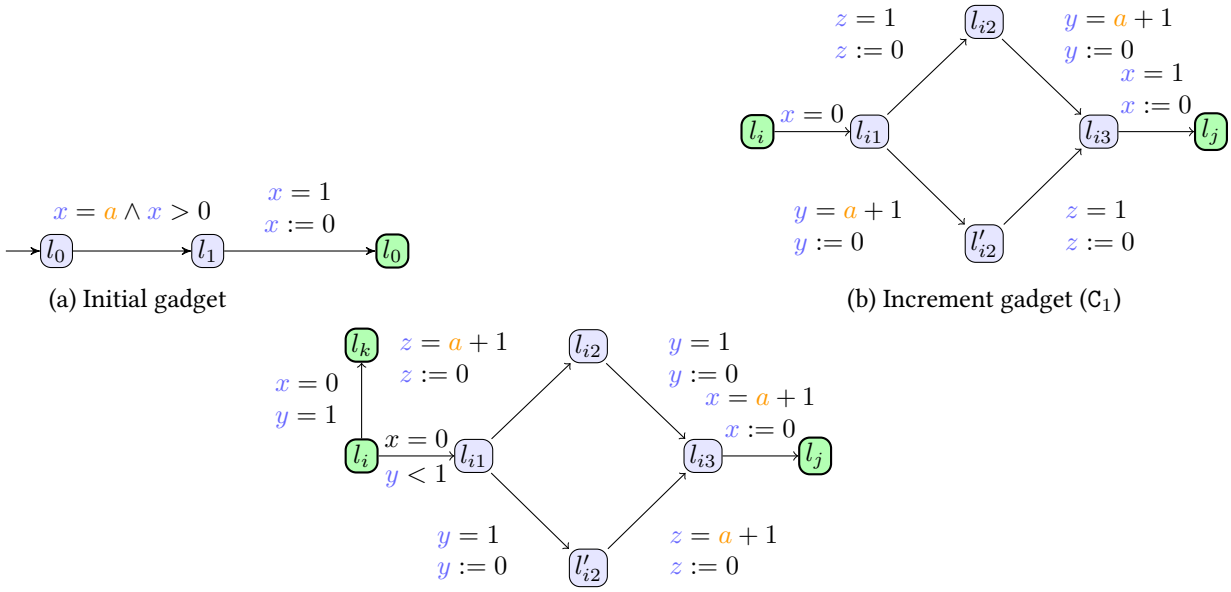
Undecidability

The EF-emptiness of PTAs is known to be undecidable, except for some bounded number of clocks and/or parametric clocks (Table 3.1). We provide below a new proof of undecidability for this problem, featuring 1 parametric clock, 3 non-parametric clocks and a single rational-valued (bounded) parameter. We reduce from the boundedness problem of a 2-counter machine, which is undecidable [Min67]. This proof is not a particularly significant advance in the state-of-the-art: this setting matches the best known proof [Mil00].

However, we will give this construction in details, because its encoding of the 2-counter machines will be used in many subsequent results in this manuscript (and was used in several papers [AM15; ALR16a; AL17a; ALR18]). This encoding of a 2-counter machine was published in [ALR16a] and was originally proposed by Didier Lime.

Recall that a deterministic 2-counter machine $\mathcal{M}$ has two non-negative counters $\mathtt{C}_1$ and $\mathtt{C}_2$, a finite number of states and a finite number of transitions, which can be of the form:

- when in state $\mathtt{q}_i$, increment $\mathtt{C}_k$ and go to $\mathtt{q}_j$;



(a) Initial gadget

(b) Increment gadget ($\mathtt{C}_1$)

(c) 0-test and decrement gadget ($\mathtt{C}_1$)

Figure 3.5 – EF-emptiness: gadgets

- when in state $\mathtt{q}_i$, if $\mathtt{C}_k = 0$ then go to $\mathtt{q}_k$, otherwise go to $\mathtt{q}_j$.

The machine starts in state $\mathtt{q}_0$ with the counters set to 0; by definition, it halts when it reaches a specific state called $\mathtt{q}_{\text{halt}}$. The boundedness problem for 2-counter machines asks whether, along the unique maximal run, the value of the counters remains smaller than some bound, and is undecidable [Min67].

Given such a machine $\mathcal{M}$, we encode it as a PTA $\mathcal{A}(\mathcal{M})$; Let us now describe this encoding in details.

Each state $\mathtt{q}_i$ of the machine is encoded as a location of the automaton, which we call l_i. The counters are encoded using clocks x, y and z and one parameter a, with the following relations with the values c_1 and c_2 of counters $\mathtt{C}_1$ and $\mathtt{C}_2$: when $x = 0$, we have $y = 1 - ac_1$ and $z = 1 - ac_2$. All three clocks are parametric[2], i. e., are compared with the parameter a in some guard or invariant of the encoding. We will see that a is a rational-valued bounded parameter, typically in $[0, 1]$ (although not bounding a has no impact on the proof).

We initialize the clocks with the gadget in Figure 3.5a (that also blocks the case where $a = 0$). Note that, in Figure 3.5, we highlight in thick green the locations of the PTA corresponding to a state of the two-counter machine (in contrast with other locations added in the encoding to maintain the matching between the clock values and the counter values). Since all clocks are initially 0, in Figure 3.5a, when in l_0 with $x = 0$, we have $y = z = 1$, which indeed corresponds to counter values 0.

We now present the gadget encoding the increment instruction of $\mathtt{C}_1$ in Figure 3.5b. The transition from l_i to l_{i1} only serves to clearly indicate the entry in the increment gadget and is done in 0 time unit. Since we use only equalities, there are really only two paths that go through the gadget: one going through l_{i2} and one through l'_{i2}. Let us begin with the former. We start from some encoding configuration: $x = 0$, $y = 1 - ac_1$ and $z = 1 - ac_2$ in l_i (and therefore the same in l_{i1}). We can enter l_{i2} (after elapsing enough time) if $1 - ac_2 \leq 1$, i. e., $ac_2 \geq 0$, which implies that $a \geq 0$, and when entering l_{i2} we have $x = ac_2$, $y = 1 - ac_1 + ac_2$ and $z = 0$. Then we can enter l_{i3} if $1 - ac_1 + ac_2 \leq 1 + a$, i. e., $a(c_1 + 1) \geq ac_2$. When entering l_{i3}, we then have $x = a(c_1 + 1)$, $y = 0$ and $z = a(c_1 + 1) - ac_2$. Finally, we can go to l_j if $a(c_1 + 1) \leq 1$ and when entering l_j we have $x = 0$, $y = 1 - a(c_1 + 1)$ and $z = 1 - ac_2$, as expected.

[2] A transformation to the setting 3 non-parametric clock + 1 parametric clock is given later on in Remark 3.1.

We now examine the second path. We can enter l'_{i2} if $1 - ac_1 \leq a + 1$, i. e., $a(c_1 + 1) \geq 0$, and when entering l'_{i2} we have $x = a(c_1 + 1)$, $y = 0$ and $z = 1 - ac_2 + a(c_1 + 1)$. Then we can go to l_{i3} if $1 - ac_2 + a(c_1 + 1) \leq 1 + a$, i. e., $a(c_1 + 1) \leq ac_2$. When entering l_{i3}, we then have $x = ac_2$, $y = ac_2 - a(c_1 + 1)$ and $z = 0$. Finally, we can go to l_j if $ac_2 \leq 1$ and when entering l_j we have $x = 0$, $y = 1 - a(c_1 + 1)$ and $z = 1 - ac_2$, as expected.

Remark that exactly one path can be taken depending on the respective order of $c_1 + 1$ and c_2, except when both are equal or $a = 0$, in which cases both paths lead to the same configuration anyway (and the case $a = 0$ is excluded by Figure 3.5a anyway).

Decrement is done similarly by replacing guards $y = a + 1$ with $y = 1$, and guards $x = 1$ and $z = 1$ with $x = a + 1$ and $z = a + 1$, respectively, as shown in Figure 3.5c. In addition, the 0-test is obtained by simply adding a transition from l_i to l_k with guard $y = 1 \wedge x = 0$, which ensures that $\mathtt{C}_1 = 0$. Similarly, the guard from l_i to l_{i1} ensures that decrement is done only when the counter is not null.

All those gadgets also work for $\mathtt{C}_2$ by swapping y and z.

The actions associated with the transitions do not matter; we can assume a single action σ on all transitions (omitted in all figures).

Lemma 3.4 ([ALR16a]). *The EF-emptiness problem is undecidable for bounded PTAs.*

Proof. We now prove that the two-counter machine $\mathcal{M}$ halts iff there exists a parameter valuation v such that $v(\mathcal{A}(\mathcal{M}))$ reaches $l_{\mathtt{halt}}$. First note that if $a = 0$ the initial gadget cannot be passed, and $l_{\mathtt{halt}}$ is reachable for no valuation. Assume $a > 0$. Consider two cases:

1. either the two-counter machine does not halt. Then, for any parameter valuation, at some point during an incrementation of, say, $\mathtt{C}_1$ we will have $a(c_1 + 1) > 1$ when taking the transition from l_{i2} to l_{i3} and the PTA will be blocked. Therefore, there exists no parameter valuation for which $l_{\mathtt{halt}}$ is reachable.

2. or the two-counter machine halts, along a (unique) run. Let c be the maximal value of this run. In that case, if $c = 0$ and $0 < a \leq 1$ or $c > 0$ and $ca < 1$, then the PTA valuated with such parameter valuations correctly simulates the machine, yielding a (unique) run reaching location $l_{\mathtt{halt}}$. The set of such valuations for a is certainly non-empty: $a = \frac{1}{2}$ belongs to it if $c = 0$ and $a = \frac{1}{c}$ does otherwise.

Hence the two-counter machine halts iff there exists a parameter valuation v such that $v(\mathcal{A}(\mathcal{M}))$ reaches $l_{\mathtt{halt}}$.

□

Remark 3.1. We can adapt our proof to fit in the most restrictive guard syntax ($x \bowtie p$) as follows: transitions with $y = a + 1$ guards and $y := 0$ reset can be equivalently replaced by one transition with a "$y = 1$" guard and a reset of some additional clock w, followed by a transition with a $w = a$ guard and the $y := 0$ reset (and similarly for x and z is the decrement gadget). This also allows the proof to work without complex parametric expressions in guards, using three additional clocks (we conjecture that a smarter encoding can be exhibited to factor these additional clocks, so as to use a single additional clock).

Decidability

The only non-trivial general class with a decidability result for EF-emptiness is L/U-PTAs [Hun+02]. We now extend this class, by proving a main positive result for IP-PTAs below:

Theorem 3.5 ([ALR16a]). *The EF-emptiness problem is decidable (and PSPACE-complete) for bounded IP-PTAs.*

Proof idea. Let $\mathcal{A}$ be a bounded IP-PTA. EF-emptiness is false for $\mathcal{A}$ iff there exists a valuation v such that a run of $v(\mathcal{A})$ reaches a location in some predefined set T. Assume there exists a valuation v such that a run of $v(\mathcal{A})$ reaches l, with $l \in T$. From [Hun+02, Proposition 3.17], there exists a symbolic run of $\mathcal{A}$ reaching a symbolic state (l, C), for some C. Since $\mathcal{A}$ is an IP-PTA, C contains at least one integer point. Hence there exists an integer parameter valuation $v' \models C{\downarrow_P}$; hence from [Hun+02, Proposition 3.18], there exists a concrete run of $v'(\mathcal{A})$ reaching l. This gives that EF-emptiness is false for $\mathcal{A}$ iff there exists an integer valuation v' such that a run of $v'(\mathcal{A})$ reaches a location in T.

Hence, deciding whether some valuation permits to reach l reduces to deciding whether some *integer* valuation permits to do so, which, for bounded PTAs, is PSPACE-complete [JLR15]. □

As any reset-PTA is an IP-PTA from Theorem 3.3, the EF-emptiness problem is decidable for reset-PTAs. Since bounded IP-PTAs are incomparable with L/U-PTAs from Theorem 3.1, and since L/U-PTAs are the only non-trivial subclass of PTAs for which the EF-emptiness problem is known to be decidable, then Theorem 3.5 strictly extends the subclass of PTAs for which this problem is decidable.

In order to get a full picture of decidability results for all subclasses of PTAs, we show the following, which comes almost directly from [Hun+02]:

Lemma 3.5 ([ALR16a]). *The EF-emptiness and EF-universality problems are decidable for closed bounded L/U-PTAs.*

Proof idea. Let $\mathcal{A}_{|bounds}$ be a closed bounded L/U-PTA. The result for EF-emptiness (resp. EF-universality) is obtained by replacing any lower-bound parameter p with $\inf(p, bounds)$ (resp. $\sup(p, bounds)$) and any upper-bound parameter p with $\sup(p, bounds)$ (resp. $\inf(p, bounds)$), and testing reachability in the obtained TA, in the spirit of [Hun+02; BL09]. □

Synthesis

Although the EF-emptiness problem is decidable for L/U-PTAs [Hun+02], the synthesis seems to pose practical problems: recall that the solution to the EF-synthesis problem for L/U-automata, if it can be computed, cannot be represented using any formalism for which emptiness of the intersection with equality constraints is decidable, which rules out the possibility of computing the solution set as a finite union of polyhedra.

We reuse the intuition of this result and extend it to non-strict bounded L/U-PTAs. Because non-strict bounded L/U-PTAs are IP-PTAs (Theorem 3.1), this result extends to (bounded) IP-PTAs too, despite decidability of the EF-emptiness problem for this latter class (Theorem 3.5).

Theorem 3.6 ([ALR16a]). *If it can be computed, the solution to the EF-synthesis problem for non-strict bounded L/U-automata and for IP-PTAs cannot be represented using any formalism for which emptiness of the intersection with equality constraints is decidable.*

3.3.2 AF-emptiness

It is known that AF-emptiness is undecidable for L/U-PTAs [JLR15]; reusing the encoding of the 2-counter machine proposed in our proof of Lemma 3.4, we now show that this result holds even for bounded L/U-PTAs. We modify the encoding by splitting the unique parameter a into a lower-bound parameter a^-

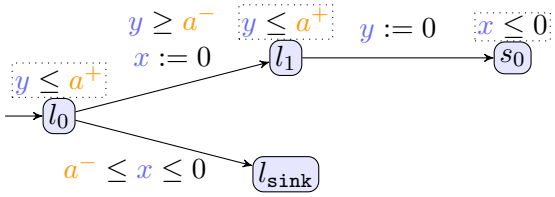


Figure 3.6 – AF-emptiness for bounded L/U-PTAs: initial gadget

and an upper-bound parameter a^+. The proof mainly relies on the gadget initializing the parameters. We initialize the parameters a^- and a^+ with the gadget in Figure 3.6 leading to the location s_0. Clearly, starting from l_0, we have $\mathsf{AF}(s_0)$ if and only if $a^- = a^+ > 0$, because

1. if $a^- = 0$ then it is possible to reach $l_{\texttt{sink}}$ and therefore we do not have $\mathsf{AF}(s_0)$, and
2. any run that reaches l_1 before y is equal to a^+ can be extended by delaying a non-null amount of time into a run that will be blocked by the invariant of s_0.

So all runs should enter l_1 with $y = a^+$, which is the case if and only if $a^- = a^+$. We therefore obtain an L/U-automaton with $a^- = a^+$ and $a^+ > 0$.

Then, the encoding is such that $l_{\texttt{sink}}$ can be reached from any location. Eventually, only if $a^- = a^+ = 0$ and only if the machine halts, then $l_{\texttt{halt}}$ is unavoidable.

As bounded L/U-PTAs are less expressive than bounded IP-PTAs and IP-PTAs, and as the proof can also work with unbounded parameters, we get a complete undecidability result for all considered subclass of PTAs:

Theorem 3.7 ([ALR16a]). *The AF-emptiness problem is undecidable for (bounded) IP-PTAs, for (bounded) L/U-PTAs and for (bounded) PTAs.*

3.3.3 EG-emptiness

Recall that the EG-emptiness problem is false if there exists at least one parameter valuation for which a maximal run remains entirely within some predefined set T of locations. That is, either this run is an infinite run, and therefore contains a cycle (remaining within T); or this run is a finite run (remaining within T), and therefore ends with a deadlock, i. e., ends with a state from which no discrete transition can be taken, even after letting some time elapse.

We will see in the following that the EG-emptiness problem stands at the frontier between decidability and undecidability for the class of L/U-PTAs. while this problem is decidable for L/U-PTAs with a bounded parameter domain with closed bounds, it becomes undecidable if either the assumption of boundedness or of closed bounds is lifted.

Theorem 3.8 ([AL17a]). *The EG-emptiness problem is decidable for closed bounded L/U-PTAs.*

Proof idea. Let $\mathcal{A}_{|bounds}$ be a closed bounded L/U-PTA and T be a subset of its locations.

The basic monotonicity property of L/U-PTAs ensures that the TA $v_{\mathrm{inf/sup}}(\mathcal{A})$, where $v_{\mathrm{inf/sup}}$ is obtained by valuating lower-bound parameters p^- by $\inf(p^-, bounds)$ and upper-bound parameters p^+ by $\sup(p^+, bounds)$, includes all the runs that could be produced with other parameter valuations. Consequently, if there is an infinite path for some valuation, there is one for $v_{\mathrm{inf/sup}}$.

In $v_{\inf/\sup}(\mathcal{A})$, it is decidable to find an infinite path staying in T, or conclude that none exist: this can be encoded into the CTL formula $\mathsf{EG}(T \wedge \mathsf{EX}T)$, to be verified on the (finite) region graph of $\mathcal{A}$ [AD94]. Since the region equivalence is a time-abstract bisimulation [TY01], this means for $\mathcal{A}$ "there exists a path that remains in T and in which every state has a discrete successor (possibly after letting some time elapse) in T". That path therefore has an infinite number of discrete actions. If we do find such a path, we can then terminate by answering yes to the EG-emptiness problem. If we do not, then in $v_{\inf/\sup}(\mathcal{A})$, all paths staying in T are finite. If we keep only discrete actions and locations, which are in finite number, the resulting paths therefore form a finite tree.

We can then explicitly compute the symbolic states (following the symbolic semantics in Definition 2.6) for all the paths in the finite tree (not only those that are maximal), and look for valuations that contain a deadlock, using techniques from e. g., [And16; AL17a].

If we find a deadlock, then we can terminate and answer yes to the EG-emptiness problem. Otherwise, we can terminate and answer no, because we have checked all the potential discrete paths staying in T for any parameter valuation. □

Now, relaxing either the boundedness or the closedness yields undecidability, as stated in the following theorems.

Theorem 3.9 ([AL17a]). *The EG-emptiness problem is undecidable for open bounded L/U-PTAs.*

Proof idea. This idea is to reuse the 2-counter machine encoding of Lemma 3.4, so that it works in constant time (1 time unit). We replace any occurrence of "1" with either a new lower-bound parameter b^- or a new upper-bound parameter b^+ (depending on whether "1" appears as a lower bound or an upper bound). We also modify the zero-test gadget so that it takes a duration strictly greater than 0, namely in $[b^-, b^+]$. We add a global invariant $w \leq 1$, where w is a fresh clock. Finally, we add a new location $l'_{\texttt{halt}}$, to which one can go from $l_{\texttt{halt}}$ iff $a^- < a^+$ or $b^- < b^+$. Therefore, provided we exclude valuations such that $a^- > a^+$ or $b^- > b^+$, there is a deadlock in $l_{\texttt{halt}}$ iff $a^- = a^+$ and $b^- = b^+$. Parameters are such that $a^-, a^+, b^+ \in [0, 1]$ while $b^- \in (0, 1]$; the interval open in 0 is essential for the proof to work.

Using an appropriate reasoning, we then have $\mathcal{M}$ halts iff $\mathsf{EG}(L \setminus \{l'_{\texttt{halt}}\})$ holds. □

Theorem 3.10 ([AL17a]). *The EG-emptiness problem is undecidable for L/U-PTAs, and therefore for PTAs.*

Proof idea. We use a reasoning similar to that of Theorem 3.9. However, we need to completely change the encoding of the 2-counter machine, and go for a discrete time encoding (that also works in dense time) inspired by a 2-counter machine encoding in [Ben+15]. The rest of the reasoning is similar to Theorem 3.9. □

3.3.4 AG-emptiness

We proved that AG-emptiness is undecidable for our new subclass of IP-PTAs. This result differentiates the classes of (bounded) L/U-PTAs and bounded IP-PTAs as AG-emptiness is decidable for (bounded) L/U-PTAs [Hun+02], and helps to understand better the boundary between decidability and undecidability for subclasses of PTAs.

Theorem 3.11 ([ALR16a]). *The AG-emptiness problem is undecidable for bounded IP-PTAs, and therefore for (bounded) PTAs.*

Proof idea. We reuse the encoding in our proof of Lemma 3.4. The main idea is, for all valuations of the parameter a that are not small enough to properly encode the counters (i. e., for some value c of a counter, $1 - ac < 0$), to allow the PTA to directly go to an $l_{\texttt{error}}$ location. In order for our encoding to be an IP-PTA (in particular the $l_{\texttt{error}}$ symbolic states), we add a new parameter b, the value of which can be typically in $[0, 1]$.

We then reduce the problem of knowing whether the counters of the machine grow unbounded along its execution,

which is undecidable [Min67], to the universality of the set of parameters that allow the encoding PTA to reach $l_{\texttt{error}}$ (i. e., EF-universality, which is equivalent to AG-emptiness). □

3.3.5 Nesting quantifiers

We show below that using *nested* quantifiers (i. e., beyond EF, EG, AF, AG) automatically leads to the undecidability, even for the very restricted class of U-PTAs with a single parameter (that can even be integer-valued).

Theorem 3.12 ([ALR18]). *The emptiness problem of a nested TCTL formula is undecidable for U-PTAs.*

Proof idea. The construction is inspired by the 2-counter machine encoding of [Ben+15], with several major adaptations. Among the many modifications (including duplicating most locations, adding transitions to a new location, etc.), the key idea is to replace guards $x = a$ with $x \leq a$, as the obtained encoding must be a U-PTA, and then use a nested TCTL formula $\mathsf{EGAG}_{=0}$ to ensure that all guards will be taken "at the latest possible moment", i. e., when $x = a$. □

We may wonder if the *timed* aspect of TCTL (and notably the urgency required by the TCTL formula $\mathsf{EGAG}_{=0}$) is responsible for the undecidability. In fact, it is not, and we could modify the proof to show that CTL itself leads to undecidability, i. e., that EGAX-emptiness is undecidable.

We conjecture the construction could be adapted so as to work over bounded time, or with a bounded parameter domain, reusing the encoding of Lemma 3.4 instead of our encoding; it should also be possible to adapt it to the dual class of L-PTAs.

Theorem 3.12 is of importance for two reasons:

- it justifies the thorough study of non-nested formulas (i. e., EF, EG, AF, AG), and
- it constitutes the first undecidable result of the literature for the class of U-PTAs, that was until then completely open (the only decidable results came from the decidable results of the larger class of L/U-PTAs, and no undecidability result was known).

A future objective will be to precisely draw the border between decidability and undecidability for U-PTAs and L-PTAs.

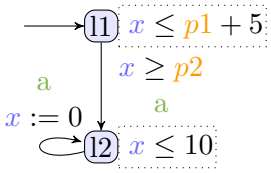
3.3.6 Cycle-existence emptiness

In contrast to deadlock-freeness that is consistently undecidable (see Section 3.3.7 below), and to EG-emptiness for which the frontier between decidability and undecidability is thin, the existence of a parameter valuation for which there exists at least one infinite run is consistently decidable for L/U-PTAs.

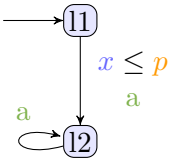
Theorem 3.13 ([AL17a]). *The cycle-existence problem is decidable for both closed bounded L/U-PTAs and for (unbounded) L/U-PTAs.*

Proof idea. The result for closed bounded L/U-PTAs derives from the following observation: in the TA obtained by valuating upper-bound (resp. lower-bound) parameters with their largest (resp. smallest) bound, there is no cycle in the zone graph iff the cycle-existence problem is true.

The result for unbounded L/U-PTAs follows from the fact that the cycle-existence problem is PSPACE-complete for integer-valued L/U-PTAs [BL09], together with a technical result [AL17a] that shows that there is a rational valuation yield-



(a) PTA deadlocked for some valuations



(b) PTA deadlocked for all valuations

Figure 3.7 – Examples of PTAs with potential deadlocks

ing a cycle iff there is an integer-valuation yielding a cycle. □

Unsurprisingly, with the rule of thumb that any non-trivial problem for PTAs is undecidable, this problem becomes undecidable for the full class of PTAs.

Theorem 3.14 ([AL17a]). *The cycle-existence problem is undecidable for PTAs.*

Proof idea. By reduction from the halting problem of a 2-counter machine, using an encoding not fundamentally different from that of Lemma 3.4. □

Remark 3.2. The reader will have noted that one subclass is not treated in this section, i. e., the class of open bounded L/U-PTAs: We conjecture that this is decidable using techniques derived from the robustness results of [San11] but the adaptation appears to require rather non-trivial developments, with techniques quite different from those we used in [AM15; ALR16a; AL17a], and is thus left to future work.

3.3.7 Deadlock-freeness emptiness and synthesis

Checking the absence of deadlocks in the model of a real-time system is of utmost importance. First, deadlocks can lead the actual system to a blockade when a component is not ready to receive any action. Second, a specificity of models of distributed systems involving time is that they can be subject to situations where time cannot elapse. This situation denotes an ill-formed model, as this situation of time blocking ("timelock") cannot happen in the actual system due to the uncontrollable nature of time.

Example 3.3. Consider the PTA in Figure 3.7a: deadlocks can occur if the guard of the transition from l_1 to l_2 cannot be satisfied (when $p_2 > p_1 + 5$) or if the invariant of l_2 is not compatible with the guard (when $p_2 > 10$). In Figure 3.7b, the system may risk a deadlock for any parameter valuation as, if the guard is "missed" (if a run chooses to spend more than p time units in l_1), then no transition can be taken from l_1.

Undecidability

The deadlock-existence emptiness problem is undecidable, even for the restricted class of closed bounded L/U-PTAs.

Theorem 3.15 ([AL17a]). *The deadlock-existence-emptiness problem is undecidable for closed bounded L/U-PTAs, for open bounded, for (unbounded) L/U-PTAs, and for PTAs.*

Proof idea. By reduction from the halting problem of a 2-counter machine, using an encoding based on Lemma 3.4. □

Synthesis

Despite the undecidability in Theorem 3.15, we can address the problem of (trying to) synthesizing valuations for which a PTA is deadlock-free. We define a semi-algorithm and, when this first procedure does not terminate, we provide a second algorithm that always terminates with an approximated result.

A semi-algorithm for deadlock-freeness synthesis First, let us introduce below our procedure PDFC, that makes use of an intermediate, recursive procedure DSynth. Both are written in a functional form in the spirit of, e. g., the reachability and unavoidability synthesis algorithms in [JLR15]. Given $\mathbf{s} = (l, C)$, we use $\mathbf{s}_C$ to denote C. The notation $g(\mathbf{s}, \mathbf{s}')$ denotes the guard of the edge from $\mathbf{s}$ to $\mathbf{s}'$. As an abuse of notation, we write $\mathsf{Succ}(\mathbf{s})$ to denote the set of successors of $\mathbf{s}$ for all possible edges.

$$\mathsf{DSynth}(\mathbf{s}, \mathsf{Passed}) = \begin{cases} \bot & \text{if } \mathbf{s} \in \mathsf{Passed} \\ \left(\bigcup_{\mathbf{s}' \in \mathsf{Succ}(\mathbf{s})} \mathsf{DSynth}(\mathbf{s}', \mathsf{Passed} \cup \{\mathbf{s}\})\right) & \\ \cup \Big(\mathbf{s}_C \setminus \left(\bigcup_{\mathbf{s}' \in \mathsf{Succ}(\mathbf{s})} \left(\mathbf{s}_C \wedge g(\mathbf{s}, \mathbf{s}')\right)^{\swarrow} \wedge \mathbf{s}'_C{\downarrow_P}\right)\Big){\downarrow_P} & \text{otherwise} \end{cases}$$

$$\mathsf{PDFC}(\mathcal{A}) = \neg \mathsf{DSynth}(s_0^{\mathcal{A}}, \emptyset)$$

First, we use a function $\mathsf{DSynth}(\mathbf{s}, \mathsf{Passed})$ to recursively synthesize the parameter valuations for which a deadlock may occur. This function takes as argument the current state $\mathbf{s}$ together with the list Passed of passed states. If $\mathbf{s}$ belongs to Passed (i. e., $\mathbf{s}$ was already met), then no parameter valuation is returned. Otherwise, the first part of the second case computes the union over all successors of $\mathbf{s}$ of DSynth recursively called over these successors; the second part computes all parameter valuations for which a deadlock may occur, i. e., the constraint characterizing $\mathbf{s}$ minus all clock and parameter valuations that allow to exit $\mathbf{s}$ to some successor $\mathbf{s}'$, all this expression being eventually projected onto P.

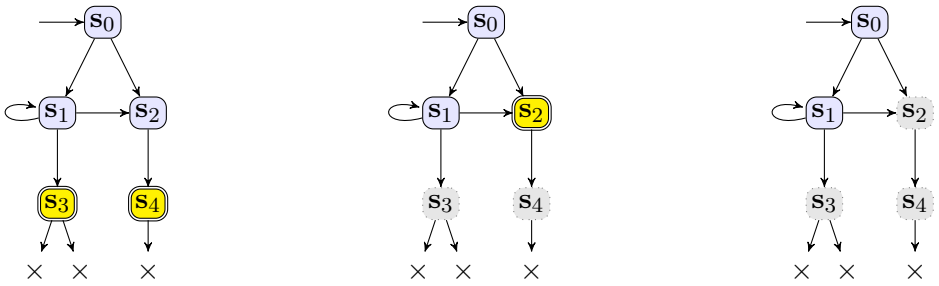
Finally, PDFC ("parametric deadlock-freeness checking") returns the negation of the result of DSynth called with the initial state of $\mathcal{A}$ and an empty list of passed states.

We show below that PDFC is sound and complete. Note however that, in the general case, the algorithm may not terminate, as DSynth explores the set of symbolic states, of which there may be an infinite number.

Proposition 3.5 ([And16]). *Assume* $\mathsf{PDFC}(\mathcal{A})$ *terminates with result* K. *Given a parameter valuation* v, $v \models K$ *iff* $v(\mathcal{A})$ *is deadlock-free.*

PDFC outputs an over-approximation of the parameter set when stopped before termination:

Proposition 3.6 ([And16]). *Fix a maximum number of recursive calls in* DSynth. *Then* PDFC *terminates and its result is an over-approximation of the set of parameter valuations for which the system is deadlock-free.*



(a) Partial state space (b) Iteration 1 (c) Iteration 2

Figure 3.8 – Application of BwUS

Under-approximated synthesis A limitation of PDFC is that either the result is exact, or it is an over-approximation when stopped earlier than the actual fixpoint. In the latter case, the result is not entirely satisfactory: if deadlocks represent an undesired behavior, then an over-approximation may also contain unsafe parameter valuations. More valuable would be an under-approximation, as this result (although potentially incomplete) firmly guarantees the absence of deadlocks in the model.

Our idea is as follows: after exploring a part of the state space in PDFC, we obtain an over-approximation. In order to get an under-approximation, we can consider that any unexplored state is unsafe, i. e., may lead to deadlocks. Therefore, we first need to negate the parametric constraint associated with any state that has unexplored successors. But this may not be sufficient: by removing those unsafe states, their predecessors can themselves become deadlocked, and so on. Hence, we proposed in [And16] a procedure BwUS that performs a backward-exploration of the state space by iteratively removing unsafe states, until a fixpoint is reached (or the constraint becomes false).

The procedure BwUS maintains several variables. Marked denotes the states that are potentially deadlocked, and the predecessors of which must be considered iteratively. Disabled denotes the states marked in the past, which avoids to consider several times the same state. K^+ is an over-approximated constraint for which there are deadlocks; the negation of K^+ is eventually returned, and therefore the algorithm returns an under-approximation. Initially, K^+ is set to the (under-approximated) result of DSynth, and all states that have unexplored successors in the state space (due to the early termination) are marked.

Example 3.4. Let us apply BwUS to a (fictional) example of a partial state space, given in Figure 3.8a. We only focus on the backward exploration, and rule out the constraint update (constraints are not represented in Figure 3.8a anyway). $\mathbf{s}_3$ and $\mathbf{s}_4$ have unexplored successors (denoted by $\times$), and both states are hence unsafe as they might lead to deadlocks along these unexplored branches.

Initially, Marked $= \{\mathbf{s}_3, \mathbf{s}_4\}$ (depicted in yellow with a double circle in Figure 3.8a), and no states are disabled. First, we add $\mathbf{s}_{3C}{\downarrow_P} \cup \mathbf{s}_{4C}{\downarrow_P}$ to K^+. Then, preds is set to $\{\mathbf{s}_1, \mathbf{s}_2\}$. We recompute the deadlock constraint for both states. For $\mathbf{s}_2$, it now has no successors anymore, and clearly we will have $\mathbf{s}_{2C}{\downarrow_P} \subseteq K^+$, hence $\mathbf{s}_2$ is marked. For $\mathbf{s}_1$, it depends on the actual constraints; let us assume in this example that $\mathbf{s}_1$ is still not deadlocked for some valuations, and $\mathbf{s}_1$ remains unmarked. At the end of this iteration, Marked $= \{\mathbf{s}_2\}$ and Disabled $= \{\mathbf{s}_3, \mathbf{s}_4\}$.

For the second iteration, we assume here (it actually depends on the constraints) that $\mathbf{s}_1$ will not be marked, leading to a fixpoint where $\mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4$ are disabled, and the constraint $\neg K^+$ therefore characterizes the deadlock-free runs in Figure 3.8c. (Alternatively, if $\mathbf{s}_1$ was marked, then $\mathbf{s}_0$ would be eventually marked too, and the result would be $\bot$.)

Experiments Both PDFC and BwUS have been implemented in IMITATOR. We ran experiments in [And16] on a set of case studies taken from the IMITATOR benchmarks library. Our benchmarks

come from teaching examples (coffee machines, nuclear plant, train controller), communication protocols (CSMA/CD [Kwi+07], RCP [CS01]), asynchronous circuits (and–or [CC05], flip-flop [CC04]), a distributed networked automation system (SIMOP [And+09a]) and a Wireless Fire Alarm System (WFAS) [Ben+15].

If an experiment has not finished within a predefined duration (in [And16], we use 300 s), the result is still a valid over-approximation; in addition, IMITATOR then runs BwUS to also obtain an under-approximation.

Analyzing the experiments, several situations occur: the most interesting result is when an exact (i. e., sound and complete) constraint is derived. For example, the constraint synthesized by IMITATOR for the PTA in Figure 3.7a is

$$p_1 + 5 \geq p_2 \wedge p_2 \leq 10,$$

which is exactly the valuation set ensuring the absence of deadlocks. In several cases, the synthesized constraint is $\bot$, meaning that no parameter valuation is deadlock-free; this may not always denote an ill-formed model, as some case studies are "finite" (no infinite behavior), typically some of the hardware case studies (e. g., flip-flop); this may also denote a modeling process purposely blocking the system (to limit the state space explosion) after some property (typically reachability) is proved correct or violated. When no exact result could be synthesized, our second procedure BwUS allows to get both an under-approximated and an over-approximated constraint (except in one case where only an over-approximation can be synthesized). This is a valuable result, as it contains valuations guaranteed to be deadlock-free, others guaranteed to be deadlocked, and a third unsure set. Full details can be found in [And16].

3.3.8 Language and trace preservation

Both the language-preservation-emptiness and the trace-preservation-emptiness problems are undecidable for PTAs, even with simple constraints in guards and invariants [AM15]. The *continuous* (or robust) versions of those problems additionally require that the language (resp. set of traces) is preserved under any intermediary valuation of the form $\lambda \cdot v + (1-\lambda) \cdot v'$, for $\lambda \in [0,1]$ (with the classical definition of addition and scalar multiplication).

Theorem 3.16 ([AM15; ALM18]). *The language-preservation-emptiness problem and its continuous version are undecidable for PTAs.*

The trace-preservation-emptiness problem and its continuous version are undecidable for PTAs.

Proof idea. The proof of the language-preservation-emptiness problem reduces from the halting problem for 2-counter machines, using an original encoding. Two clocks are used to encode the value of the two counters, while a third clock is used to count modulo p (the only, integer-valued, parameter). A fourth clock is specifically used to block the computation after p time units, and is used by the language-preservation-emptiness problem.

The proof of the trace-preservation-emptiness problem is rather technical, and comes with three flavors:

1. the first proof involves diagonal constraints (i. e., of the form $x_i - x_j \bowtie plt$, which goes beyond the syntax of PTAs), but only a fixed number of parametric clocks [AM15];
2. the second proof does not involve diagonal constraints. It involves a bounded number of locations (but with an unbounded number of transitions) and an unbounded number of parametric clocks; by unbounded we mean not constant but depending on the size of the counter-machine [AM15];
3. the third proof uses a bounded number of clocks and parameters, and an unbounded number of locations [ALM18].

□

The need for an unbounded number of clocks in the first two versions of this proof comes from the fact that the proof encodes the 2-counter machine with a fixed number of locations (to reduce easily from

language-preservation to trace-preservation), which thus requires to encode each location with a different clock. Note that the first two versions of the proof are, to the best of our knowledge, the only attempt to model a 2-counter machine using PTAs with a constant number of locations (at the cost of an unbounded number of clocks).

The language-preservation-emptiness problem is also undecidable for L/U-PTAs.

Theorem 3.17 ([AM15]). *The language-preservation-emptiness and trace-preservation-emptiness problems and their continuous versions are undecidable for L/U-PTAs.*

Proof idea. The construction reuses the reasoning used in Theorem 3.16, with an additional initial gadget: we split the parameter p into an upper-bound and a lower-bound parameter, and force them to be equal to each other to preserve the language of a given parameter valuation. □

What's beyond...? Most of the results of [AM15] were initially defined for a different version of the language: the language is defined as the set of untimed words associated with *all* maximal runs (without any acceptance condition), i. e., all deadlock runs and all infinite runs. However, we show in [ALM18] that both these results extend in a straightforward manner to the definition in Definition 2.3.

In addition, we show in [ALM18] that looking for valuations for which the language (or trace set) is included / is strictly included / includes / strictly includes the language (or trace set) of the reference valuation preserves the undecidability of the problem.

Finally, the results remain valid even over bounded-time [ALM18].

The language and trace preservation can therefore seen as consistently undecidable.

Our undecidability results can be put into perspective with the decidability results for the larger class of hybrid automata of [Bri+13]. In [Bri+13], time-bounded reachability is proved decidable for a subclass of hybrid automata with monotonic (either non-negative or non-positive) rates: parametric timed automata can fit into this framework: clocks and parameters all have non-negative rates (1 for clocks, and 0 for parameters). To "initialize" parameters, one can initialize them to 0, let time elapse for an arbitrary amount of time (for each parameter), and then set their rate to 0 (while resetting all clocks). However, to compare clocks and parameters together in a hybrid automaton, one needs diagonal constraints—that are not allowed in [Bri+13]. As we showed that our undecidability results hold over bounded-time with a single parameter, one can revisit the result of [Bri+13] as follows: allowing a single variable (our parameter) in diagonal constraints, with only one location with a non-zero rate for this variable (the initialization location for this parameter) renders the decidable problem of [Bri+13] undecidable.

In addition, we can compare the undecidability of the language-preservation-emptiness problem with some related results.

In timed automata, the (untimed) *language robustness* problem asks whether there exists a $\Delta > 0$ for which the TA where all guards are enlarged by Δ gives the same untimed language as the original TA. The result is decidable [San11] for some assumptions: beyond some mild restrictions (closed guards and bounded clocks), this work requires the *progress cycle* assumption, i. e., that any cycle in the structural automaton resets all clocks at least once. The complexity is 2EXPTIME (under these assumptions) and PSPACE with the additional requirement of deterministic TAs.

In [SBM14], the shrinkability problem is considered: instead of enlarging guards, the existence of a positive constant (possibly different for each guard) such that the TA where each guard is *shrinked* by this constant time-abstract bisimulates the original TA is decidable.

In time Petri nets, the robust untimed language preservation is studied in [Aks+16]: "given a bounded TPN N, does there exist a $\Delta > 0$ such that the untimed language of N is equal to the untimed language

Class	U-PTAs	bL/U-PTAs		L/U-PTAs	bIP-PTAs	IP-PTAs	bPTAs	PTAs
		closed	open					
EF	[Hun+02]	[ALR16a]	open	[Hun+02]	[ALR16a]	*[ALR16a]*	*[Mil00]*	*[AHV93]*
AF	open	*[ALR16a]*		*[JLR15]*	*[ALR16a]*	*[ALR16a]*	*[ALR16a]*	*[JLR15]*
EG	open	[AL17a]	*[AL17a]*	*[AL17a]*	open		*[AL17a]*	
AG	[AL17a]	[ALR16a]	open	[AL17a]	*[ALR16a]*			
TCTL	*[ALR18]*	*[ALR16a]*		*[JLR15]*	*[ALR16a]*		*[Mil00]*	*[AHV93]*
EC	[AL17a]	[AL17a]	open	[AL17a]	open		*[AL17a]*	
ED	open	*[AL17a]*			open		*[AL17a]*	*[And16]*
LgP	open	*[AM15]*			open		*[AM15]*	
TrP	open	*[AM15]*			open		*[AM15]*	

Table 3.2 – Decidability of the emptiness problems for PTAs and subclasses

of N where each firing interval is enlarged by Δ"? This problem is undecidable in general [Aks+16], but becomes decidable for *distinctly labeled bounded TPNs*, i. e., when all transitions are labeled with different labels. The result comes from a translation from TAs to TPNs and the decidability of the robustness of timed automata w.r.t. ω-regular properties [BMS11]. The setting of distinctly labeled transitions was not considered in [AM15], and would deserve investigation too.

3.3.9 The one-clock case

In [AM15], we show that the parametric zone graph $\mathcal{PZG}$ is finite for a single (parametric) clock and arbitrarily many rational-valued parameters over dense time. This result is not fundamentally new: in [AHV93], using a dynamic programming fashion procedure, the set of all disjunctive path constraints can be computed, from which emptiness can be inferred.[3] However, our result extends to rational-valued parameters and, most importantly, is redefined for the case of the zone graph, that is manipulated by many synthesis algorithms in the literature. This implies that all problems that reason on the zone graph can be decided; in addition, exact synthesis can be achieved. This includes in particular EF-, EG-, AF and AG-emptiness: two semi-algorithms based on the zone graph are proposed in [JLR15] that compute (if they terminate) all valuations for EF-synthesis and AF-synthesis respectively. Since the zone graph is finite and both procedures are semi-algorithms, they output an exact result for EF- and AF-synthesis in the form of a finite union of polyhedra. From these results, one can trivially derive not only EF- and AF-emptiness, but also EG- and AG-emptiness since they are equivalent to AF- and EG-universality respectively.

The language- and trace-preservation-emptiness problems are decidable for deterministic PTAs with a single (parametric) clock, and with linear parameter constraints allowed in guards and invariants, i. e., of the form $x \bowtie plt$ or $plt \bowtie 0$ [AM15]. The procedure IM (recalled in Algorithm 2 in Section 2.4) synthesizes (when it terminates) all parameter valuations with the same trace set as a given valuation, that is complete only for deterministic PTAs, and terminates in the case of a single clock.

3.3.10 Summary of decision problems

We give a summary in Table 3.2. We give from left to right the (un)decidability for U-PTAs, bounded L/U-PTAs (with either closed or open bounds), bounded IP-PTAs, L/U-PTAs, IP-PTAs, bounded PTAs, and PTAs. We review the emptiness of TCTL subformulas (EF, AF, EG, AG), full TCTL, cycle-existence, deadlock-

[3] In fact, the result in [AHV93] even extends the decidability to a single parametric clock with arbitrarily many non-parametric clocks. This is not the case of our result.

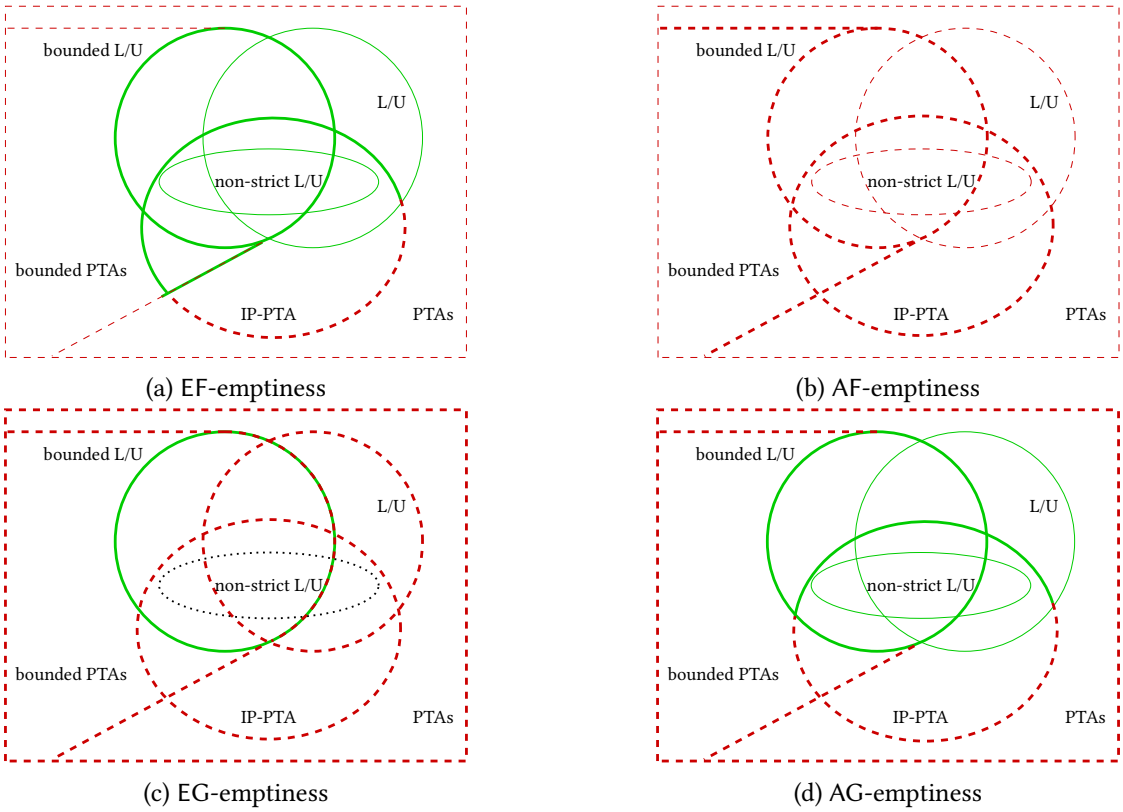


Figure 3.9 – Decidability results for PTAs and subclasses

existence and language- and trace-preservation. Decidability is given in green, whereas undecidability is given in italic red. Our contributions are emphasized in bold using a plain background, whereas existing results are depicted using a light background. When several papers in the literature proved the same result, we only give the earliest result, and not necessarily the best (in terms of number of clocks and parameters, or complexity). For EF-emptiness, a full table with all best results was given in Table 3.1.

We give another summary in Figure 3.9. Recall that bounded L/U-PTAs and L/U-PTAs are in fact incomparable; they are therefore not included into each other in the figures. Decidability (resp. undecidability) is depicted in plain green (resp. dashed red); open problems are depicted in dotted black. Our contributions are depicted in thick.

What's open? Beyond the open problems for IP-PTAs (that are mainly open because we did not study them), we discuss in the following the interesting open problems. I highly suspect that EF-emptiness and AG-emptiness (in fact EF-universality) are decidable for open bounded L/U-PTAs: note that decidability holds both for closed bounded and unbounded L/U-PTAs. In both cases, it must be possible to consider the TA valuated with the parameter (possibly open) bounds: EF-emptiness could be solved by replacing non-strict inequalities with strict inequalities, while AG-emptiness could be solved by studying the region graph [AD94]. This is the subject of ongoing work.

I suspect that robustness results of [SBM14] might be used to prove the decidability of the EC-emptiness problem for open bounded L/U-PTAs; however, preliminary discussions with Didier Lime and Ocan Sankur showed that this involves some technicalities—and will be the subject of future work.

Finally, results on U-PTAs are highly challenging, and will make an interesting future work.

3.4 Perspectives

Expressiveness A comparison of the expressiveness of these different syntactic models remains to be done. Whereas it is likely that allowing constraints of the form $x \bowtie plt$ may be simulated using constraints of the form $x \bowtie p \mid d^+$ (perhaps adding additional locations, clocks and parameters), the expressiveness may differ when adding a set of accepting locations (just as the timed expressive power of TSA is strictly less than that of TBA [HKW95]).

The question of the relationship between syntax and expressiveness: allowing a richer syntax than $x \bowtie p$ in guards and invariants (e. g., $x \bowtie p + 1$) may have an impact on the expressiveness, at least in terms of numbers of clocks and parameters; and since the number of clocks has an impact on decidability, this is to be studied. For example, in [ALR16b], we showed that allowing fully parametric constraints does not change the expressiveness of parametric timed automata, but at the cost of a larger number of clocks and parameters. An interesting direction of research would be to define an equivalence such as "$x \bowtie p + 1$ is equivalent to $x \bowtie p$ at the cost of one extra clock".

Our definitions of expressiveness only hold for integer-valued parameters. Extending them to rational-valued parameters is interesting (and will have an impact, for example for bounded PTAs).

Two open and promising formalisms Exhibiting the precise border between decidability and undecidability for U-PTAs and L-PTAs is of interest, as these classes remain largely open, while still allowing expressive models. In fact, they can somehow be considered as the simplest extension of timed automata with timing parameters.

Beyond parametric timed automata Our syntactic class of reset-PTA seems promising, especially augmented with some features. Mathias Ramparison's PhD work focuses on an extension of reset-PTAs for which the EF-emptiness problem remains decidable. Also combining this restriction with the restriction of *initialization* in hybrid automata [Hen+98] seems of interest.

Chapter 4

Efficient verification

After studying theoretical issues on parametric timed automata in Chapter 3, we summarize here more practical contributions, that all aim at making synthesis more efficient. These can be seen as *pragmatic* contributions: since most problems are undecidable, let us define optimizations with the hope that they will help most case studies terminating.

We first adapt to parametric timed automata two existing techniques defined for timed automata, i. e., convex state merging (Section 4.1, with Laurent Fribourg and Romain Soulat) and dynamic clock elimination (Section 4.2).

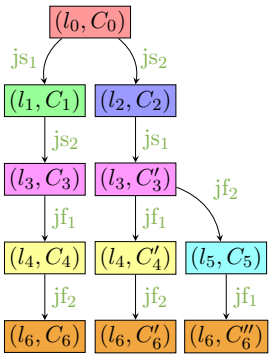
Then, we show in Section 4.3 (with Didier Lime and Olivier Roux) that the notion of integer hull, proposed in [JLR15] to perform symbolic synthesis for bounded integer-valued parameters, can be used to ensure termination of algorithms returning a rational-valued, i. e., *dense*, result; while completeness may be lost, we do keep the integer-completeness.

We then use distributed computing techniques (running on a cluster, i. e., on a set of computers with their own memory and linked by a network) to perform parameter synthesis (Section 4.4, with Camille Coti, Nguyễn Hoàng Gia and Sami Evangelista). Distributing verification has been addressed in several earlier works, e. g., statistical model checking [Bul+11], or verification of probabilistic [Ces+16] or timed systems [ZNL16a; ZNL16b]. Also, multi-core LTL verification [Eva+12] and emptiness checking of timed Büchi automata [Laa+13] was considered; but these run on multicore computers (with a shared memory) whereas our primary goal is to run verification on a cluster (where each node has its own memory). In fact, to the best of our knowledge, our work is the first attempt to distribute parameter synthesis, particularly in parametric timed formalisms.

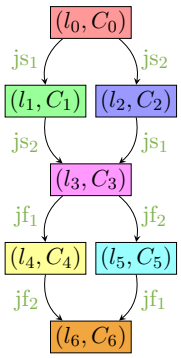
We then propose a new algorithm that preserves the reachability of a location for a given valuation, and show that, by performing repeated applications on different valuations, it can outperform the classical EFsynth, especially when distributed (Section 4.5, with Giuseppe Lipari and Sun Youcheng).

Finally, we use learning-based techniques for (non-)parametric synthesis, and show that in a syntactic subclass of PTAs, namely parametric event-recording automata, we can outperform EFsynth by an order of magnitude using compositional verification (Section 4.6, with Lin Shang-Wei).

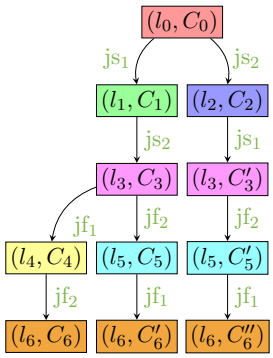
IMITATOR Most of these contributions (as well as those of Chapters 5 and 6) have been implemented in IMITATOR [And+12], a tool supporting (extensions of) parametric timed automata as an input language. I initiated the development of IMITATOR in 2008 (during my PhD thesis) and, although I implemented most algorithms in IMITATOR, some of my colleagues (namely Camille Coti, Sami Evangelista, Nguyễn Hoàng Gia and Romain Soulat) took part to some development too.



(a) Trace set of $v(\mathcal{A})$



(b) Symbolic semantics



(c) Trace set of $v'(\mathcal{A})$

Figure 4.1 – Trace sets of $\mathcal{A}$

4.1 Convex state merging

In order to fasten the computation of the symbolic state space of PTA according to Definition 2.6, we can *merge* states together using *convex state merging*: if two states share the same location, and the union of their continuous part is included into their convex hull, then the two states can be replaced with their hull.

In [BBM06], it is shown that, in a network of TAs, all the successor states can be merged together when all the interleavings of actions are possible. However, this result does not seem to extend to the parametric case. In [Dav05; Dav06], it is proposed to replace the union of two states by a unique state when the union of their continuous part (i. e., the symbolic clock values) is convex, and the discrete part (i. e., the location) is identical. This technique is applied to timed constraints represented in the form of Difference Bound Matrices (DBMs) (see e. g., [BY03]).

Here, we attempt to extend the merging described in [Dav05; Dav06] to the parametric case. This extension is not trivial, and the implementation is necessarily different, since DBMs (in their original form) do not allow the use of parameters. Instead, we implemented our approach in IMITATOR using polyhedra [BHZ08].

Definition 4.1 (Merging)**.** Two states (l_1, C_1) and (l_2, C_2) are *mergeable* if $l_1 = l_2$ and $C_1 \cup C_2$ is convex; then, $(l_1, C_1 \cup C_2)$ is their *merging*.

We showed in [AFS13a] that, while convex state merging can be used to improve the efficiency of EFsynth, it cannot be used for the inverse method (recalled in Section 2.4).

Example 4.1. We use here a typical jobshop example in the setting of parametric schedulability [Fri+12], in order to show that the traces are no longer preserved with the inverse method when state merging is used. This system (modeled by a PTA $\mathcal{A}$) contains 2 machines on which 2 jobs should be performed. The system parameters are p_i (for $i = 1, 2$) that encode the duration of each job. The system actions are js_1 (job 1 starting), jf_1 (job 1 finishing) and similarly for job 2.

Consider $v = \{p_1 \leftarrow 1, p_2 \leftarrow 2\}$. The trace set of $v(\mathcal{A})$ is given in Figure 4.1a (in the form of a graph). Applying the inverse method to $\mathcal{A}$ and v gives $K = p_2 > p_1$. From the correctness of the inverse method [And+09b], the trace set of $v(\mathcal{A})$, for all $v' \models K$, is the same as for $v(\mathcal{A})$. Now, applying the inverse method to $\mathcal{A}$ and v while merging states on-the-fly gives $K' = \top$; the parametric reachability graph built using state merging is given in Figure 4.1b. Then, let $v' = \{v_1 \leftarrow 2, v_2 \leftarrow 1\}$ be a valuation in K' but outside of K. The trace set of $v'(\mathcal{A})$ is given in Figure 4.1c. The trace sets of $v(\mathcal{A})$ and $v'(\mathcal{A})$ are different: the trace

$l_0 \overset{\text{js}_2}{\Rightarrow} l_2 \overset{\text{js}_1}{\Rightarrow} l_3 \overset{\text{jf}_1}{\Rightarrow} l_4 \overset{\text{jf}_2}{\Rightarrow} l_6$ exists in $v(\mathcal{A})$ but not in $v'(\mathcal{A})$; the trace $l_0 \overset{\text{js}_1}{\Rightarrow} l_1 \overset{\text{js}_2}{\Rightarrow} l_3 \overset{\text{jf}_2}{\Rightarrow} l_5 \overset{\text{jf}_1}{\Rightarrow} l_6$ exists in $v'(\mathcal{A})$ but not in $v(\mathcal{A})$. Note that the reachable locations and executable actions are the same in these two trace sets.

However, in general, using convex merging may even not preserve the set of "reachable actions", i. e., actions belonging to a run of the original TA.

Nevertheless, we proved that merging still preserves the set of reachable locations in the framework of the language or trace-preservation-synthesis.

Theorem 4.1 ([AFS13a]). *Let K be the result of the inverse method applied to a PTA $\mathcal{A}$ and a parameter valuation v while merging states on-the-fly. Then, for all $v' \models K$, $\mathsf{ReachLocs}(v'(\mathcal{A})) = \mathsf{ReachLocs}(v(\mathcal{A}))$.*

What's beyond...? In fact, we can improve these theoretical results in two ways [AFS13a]:

1. By adding the assumption of backward-determinism, i. e., for any location, at most one action is used on its incoming edges (which can be checked syntactically), then the set of actions is preserved too.
2. By proposing a variant of the combination of the inverse method with the merging (by changing the time where merging is performed), we again retrieve the preservation of the set of actions. This comes at the cost of a less efficient algorithm in practice according to our experiments.

Convex merging in practice We conducted experiments in [AFS13a] comparing the efficiency of merging w.r.t. the traditional algorithms. For the inverse method, and for reachability analysis, despite the very high cost of the mergeability test (performed in IMITATOR by PPL [BHZ08]), the improvement is often dramatic, especially for parametric schedulability analyses. This comes from the fact that, due to merging, the number of symbolic states is decreased by an order of magnitude, leading to a much more efficient inclusion check when new symbolic states are computed. This led us to set as default convex merging in most algorithms in IMITATOR, when the equality of trace sets is not required.

4.2 Dynamic clocks elimination

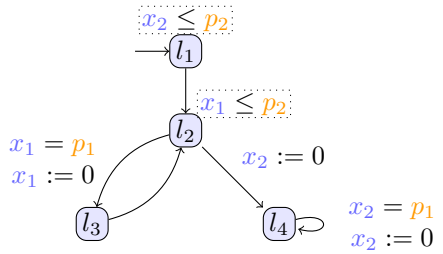
It is well known that the fewer clocks, the more efficient real-time model checking is [BY03]. Furthermore, a smaller number of clocks may imply a more compact state space: when constraints are represented using arrays and matrices, the fewer clocks, the smaller the constraints are, the more compact the state space is. Formalisms such as (parametric) timed Petri nets [TLR09] or (parametric) stateful timed CSP [Sun+13a; And+14] have the advantage to dynamically create and discard clocks (called "firing times" in Petri nets). Hence, clocks only appear in symbolic states when they are actually useful. In contrast, in (parametric) timed automata, according to their standard semantics, clocks must be present in all states.

In [DY96], two methods are proposed to reduce the number of clocks in TAs:

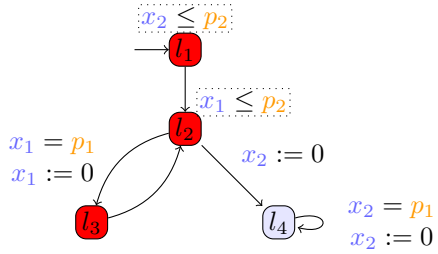
1. the detection of active clocks (the other clocks can be safely eliminated), and
2. the detection of clocks equal to each others (in which case only one such clock can be kept).

It is shown that the resulting automaton is bisimilar to the original one, and experiments show large state space reductions.

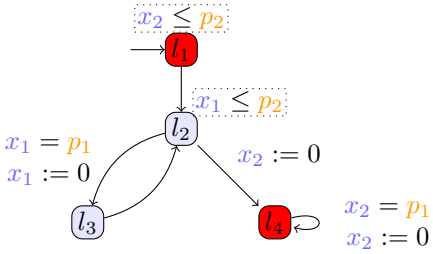
We extended the first method to the parametric case. However, there are technicalities: in [DY96], the constraints are implemented in the form of difference bound matrices, where adding and removing clocks



(a) A toy PTA $\mathcal{A}$



(b) Not marked in red: locations where x_1 is irrelevant



(c) Not marked in red: locations where x_2 is irrelevant

Figure 4.2 – Static computation of the useless clocks: an example

is straightforward. In contrast, we use polyhedra in IMITATOR, where such operations are much more costly. Finally, our original motivation was to ensure termination of some systems, which is not necessary in the non-parametric setting since most algorithms rely on symbolic state space partitions guaranteeing termination.

We introduced in [And13a] a technique to eliminate clocks on-the-fly, when it is guaranteed that they will not be read in guards and invariants until their next reset. Our approach is based on a static computation of the locations where clocks can be safely eliminated, as well as on a dynamic elimination of these clocks during the analysis.

Example 4.2. Consider the toy PTA $\mathcal{A}$ in Figure 4.2a. Our aim is to detect in which locations which of the two clocks x_1 and x_2 can be safely eliminated, because its current value does not have impact on the analysis, nor its future—until the next reset of that clock. The results of the application of our static procedure to $\mathcal{A}$ and to x_1 and x_2 are given in Figures 4.2b and 4.2c, respectively. The only location for which x_1 is useless is l_4, while the locations for which x_2 is useless are l_2 and l_3.

In the case of a network of PTA (i. e., several PTAs in parallel, synchronizing using shared actions, as allowed by the input syntax of IMITATOR), the list of useless clocks in a global location is the union, for each of the PTA in parallel, of the clocks useless in the local location for this PTA.

Dynamic clocks elimination in practice We implemented our approach in IMITATOR. Experiments tabulated in [And13a] show a diminution of the number of states and of the computation time, and in some cases allow termination of the analysis of models that could not terminate otherwise. Surprisingly, even when the number of clocks (and hence of states) remains constant, the computation time does not increase, i. e., there is little noticeable overhead in applying the proposed clock elimination. This tends to show that the dynamic elimination of clocks should become default in IMITATOR.

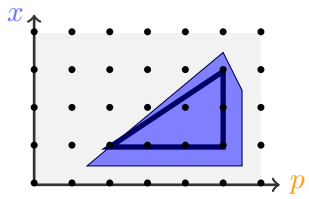


Figure 4.3 – Illustration of the integer hull

4.3 Guaranteeing termination with the integer hull

4.3.1 Context and objective

In [JLR15], a focus is made on integer-valued bounded parameters (still over dense-time), for which many problems are obviously decidable (in fact PSPACE-complete). The authors provided symbolic algorithms to compute the exact set of correct integer parameter values ensuring a reachability (EF) or unavoidability (AF) property, using two ad-hoc algorithms called IEFsynth and IAFsynth, respectively. These algorithms are shown to be more efficient in practice than an exhaustive enumeration with Uppaal. A drawback is that returning only integer points prevents designers to use the synthesized constraint to study the robustness or implementability of their system (in the sense of e. g., [De +08; Mar11; BMS13]).

IEFsynth is a classical synthesis algorithm that traverses the symbolic semantics of a PTA (Definition 2.6), with a main exception: when finding a new symbolic state, instead of comparing its constraint with the constraint of already visited states, it compares their *integer hull*. Let us recall this notion.

Integer hulls Let C be a convex polyhedron. C is topologically closed if it can be defined using only non-strict inequalities.[1]

Definition 4.2 (integer hull)**.** The *integer hull* of a topologically closed polyhedron, denoted by $\mathsf{IH}(C)$, is defined as the convex hull of the integer vectors of C, i. e., $\mathsf{IH}(C) = \mathsf{Conv}(\mathsf{IV}(C))$, where Conv denotes the convex hull, and IV the set of vectors with integer coordinates.

Example 4.3. Consider the polyhedron with a blue background in Figure 4.3. Then its integer hull is the polyhedron delimited with a black thick line; observe that it contains all integer points of the original polyhedron.

IEFsynth is recalled in Algorithm 3. It differs from EFsynth (recalled in Algorithm 1) with only two points (highlighted in Algorithm 3):

1. the *integer hulls* of the symbolic states are compared to each other (lines 4 and 6), and
2. the projection onto the parameters of the *integer hull* of the target states is returned (line 1).

Whereas EFsynth does not necessarily terminate, IEFsynth terminates with an exact result interpreted over integer parameter valuations.

[1] We only define here the integer hull of a topologically closed polyhedron. In fact, any non-closed polyhedron can be represented by a closed polyhedron with one extra dimension [HPR94]. Direct handling of not-necessarily-closed (NNC) polyhedra raises no theoretical issue but would impair the readability of this manuscript (see [JLR15] for details).

Algorithm 3: IEFsynth$(\mathcal{A}, s, T, S)$

input : A PTA $\mathcal{A}$, a symbolic state $s = (l, C)$, a set of target locations T, a set S of passed states on the current path

output: Constraint K over the parameters

```
1 if l ∈ T then K ← (IH(C))↓P ;
2 else
3     K ← ⊥;
4     if IH(s) ∉ S then
5         for each outgoing edge e from l in A do
6             K ← K ∪ IEFsynth(A, Succ(s, e), T, S ∪ {IH(s)});
7 return K
```

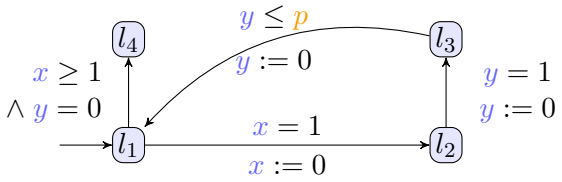


Figure 4.4 – A PTA motivating the need for an extrapolation

Proposition 4.1 ([JLR15]). *Given a bounded PTA, IEFsynth always terminates. In addition, given an integer parameter valuation v, $v \models$ IEFsynth$(\mathcal{A}, s_0, T, \emptyset)$ iff T is reachable in $v(\mathcal{A})$.*

A natural question follows: since the result of IEFsynth comes in the form of a convex constraint, can this convex constraint be interpreted over the set of rationals (instead of integers)?

4.3.2 A parametric extrapolation

First, let us motivate the use of an extrapolation.

Example 4.4. Consider the PTA in Figure 4.4. The duration between any two resets of x is 1, while the duration between any two resets of y is in $[1, 1 + p]$, which requires p to be arbitrarily small to do arbitrarily many loops through $l_1 l_2 l_3$. In fact, assuming $p \in [0, 1]$, the (desired) answer to the EF-synthesis with l_4 as the goal location would be $0 < p \leq 1$. After a number n of times through the loop, we get constraints in l_1 of the form $0 \leq x - y \leq n \times p$, with n growing without bound. Even if the parameter p is bounded (e. g., in $[0, 1]$), the time necessary to reach location l_4 is unbounded. This was not the case in [JLR15] due to the fact that parameters were bounded integers. Hence, on this PTA, we cannot just apply the integer hull (as in [JLR15]) to ensure termination and integer-completeness of our algorithms.

Let us now show that the union for all values of the parameters of the classical k-extrapolation used for the zone-abstraction for timed automata (see e. g., [Beh+06]) leads to a non-convex polyhedron.

Example 4.5. Let us consider the PTA in Figure 4.5a with a parameter p such that $0 \leq p \leq 1$. By taking n

times the loop we obtain:

$$0 \leq x \leq p \ \wedge \ 0 \leq y - x \leq (n+1) \times p \ \wedge \ 0 \leq p \leq 1$$

The greatest constant of the model is $k = 1$. After one loop, y can be greater than 1. Then, for each value of p, we can apply the classical k-extrapolation used for timed automata (as recalled in [Beh+06]) of the corresponding zone. The union for all values of p of these extrapolations, projected to the plan (y, p) is depicted by the plain blue part (light and dark blue) of Figure 4.5b. The obtained polyhedron is non-convex.

(a) PTA (assume $p \in [0, 1]$)

(b) Extrapolation

Figure 4.5 – Example illustrating the non-convex parametric extrapolation

For any zone C and variable x, we denote by $\mathsf{Cyl}_x(C)$ the *cylindrification* of C along variable x, i. e., $\mathsf{Cyl}_x(C) = \{w \mid \exists w' \in C, \forall x' \neq x, w'(x') = w(x') \text{ and } w(x) \geq 0\}$. This is a usual operation that consists in *unconstraining* variable x.

Definition 4.3 ((M, x)-extrapolation)**.** Let C be a polyhedron. Let M be a non-negative integer constant and x be a clock. The (M, x)-extrapolation of C, denoted by $\mathsf{Ext}_x^M(C)$, is defined as:

$$\mathsf{Ext}_x^M(C) = \big(C \cap (x \leq M)\big) \cup \mathsf{Cyl}_x\big(C \cap (x > M)\big) \cap (x > M).$$

Given $s = (l, C)$, we write $\mathsf{Ext}_x^M(s)$ for $\mathsf{Ext}_x^M\big(C\big)$.

Example 4.6. To illustrate the (M, X)-extrapolation, we go back to the example of Figure 4.5a after one loop. C is the polyhedron for $n = 1$. $\mathsf{Ext}_y^1(C)$ is depicted in Figure 4.5b by the plain blue part as follows: $\big(C \cap (y \leq 1)\big)$ is in light blue and $\mathsf{Cyl}_y\big(C \cap (y > 1)\big) \cap (y > 1)$ is in dark blue.

We can now consistently define the (M, X)-extrapolation operator:

Definition 4.4 ((M, X)-extrapolation)**.** Let M be a non-negative integer constant and X be a set of clocks. The (M, X)-extrapolation operator Ext_X^M is defined as the composition (in any order—which we showed in [ALR15] to be equivalent) of all Ext_x^M, for all $x \in X$.

4.3.3 Ensuring termination of parameter synthesis

Using our extrapolation, we will now ensure termination of IEFsynth even for rational-valued parameters. The modified version, called RIEFsynth ("R" stands for "robust") is given in Algorithm 4, and differs from IEFsynth in two points:

Algorithm 4: RIEFsynth$(\mathcal{A}, s, T, S)$

input : A PTA $\mathcal{A}$, a symbolic state $s = (l, C)$, a set of target locations T, a set S of passed states on the current path

output: Constraint K over the parameters

1 **if** $l \in T$ **then** $K \leftarrow C{\downarrow_P}$;
2 **else**
3 $\quad K \leftarrow \bot$;
4 $\quad$ **if** $\mathsf{IH}(\mathit{Ext}^M_X(s)) \notin S$ **then**
5 $\quad\quad$ **for** *each outgoing edge* e *from* l *in* $\mathcal{A}$ **do**
6 $\quad\quad\quad K \leftarrow K \cup \mathsf{RIEFsynth}\big(\mathcal{A}, \mathsf{Succ}(s, e), T, S \cup \{\mathsf{IH}(\mathsf{Ext}^M_X(s))\}\big)$;
7 **return** K

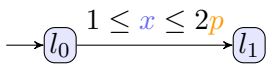


Figure 4.6 – A PTA for which RIEFsynth synthesizes a more complete result than IEFsynth

1. contrarily to IEFsynth, RIEFsynth does not return the projection *of the integer hull* of the target constraint, but the projection of the *full constraint* C (line 1); and
2. RIEFsynth compares integer hulls of *extrapolations of* constraints (lines 4 and 6).

Example 4.7. Consider the PTA in Figure 4.6 (assume the unique goal location is l_1). To ensure the $\mathsf{EF}\{l_1\}$ property, we just need to be able to go through the transition from l_0 to l_1. The parametric zone C_1 obtained in l_1 is $1 \leq x \wedge 1 \leq 2p$, which implies $p \geq \frac{1}{2}$. The integer hull of C_1 is $1 \leq x \wedge 1 \leq p$, which implies $p \geq 1$.

Algorithm IEFsynth gives the result $p \geq 1 \wedge p \in \mathbb{N}$, while algorithm RIEFsynth gives (here) the exact result $p \geq \frac{1}{2}$.

As there is a finite number of integer hulls of extrapolations of symbolic states [ALR15], RIEFsynth explores only a finite number a symbolic states.

Theorem 4.2 ([ALR15]). *For any bounded PTA $\mathcal{A}$, the computation of RIEFsynth$(\mathcal{A}, s_0, T, \emptyset)$ terminates.*

Upon termination of RIEFsynth, we have:

1. *Soundness: If $v \in$ RIEFsynth$(\mathcal{A}, s_0, T, \emptyset)$ then T is reachable in $v(\mathcal{A})$.*
2. *Integer completeness: If v is an integer parameter valuation, and T is reachable in $v(\mathcal{A})$ then $v \in$ RIEFsynth$(\mathcal{A}, s_0, T, \emptyset)$.*

Observe that our method allow us to obtain a result *arbitrarily precise*: by rescaling the constants, instead of synthesizing (at least) all integer-points, we can also synthesize all points multiple of an arbitrarily small value (e. g., 10^{-n}, with n as large as desired). Also note that, contrarily to discrete model checking, arbitrarily large constants have an impact near-to-null in parametric model checking (the only impact is the management of these large constants in exact arithmetics, which remains almost negligible in implementations such as IMITATOR when compared to the high cost of other operations).

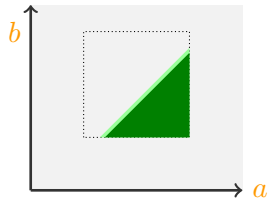


Figure 4.7 – Comparison of the results of IEFsynth and RIEFsynth

4.3.4 Implementation

The integer hull was not implemented in IMITATOR, but an implementation in Roméo was performed by Didier Lime. Polyhedra operations (both convex and non-convex) are handled by the PPL library [BHZ08]. To illustrate this, we refer the reader to the scheduling example of [JLR15]. It consists in three tasks t_1, t_2, t_3 scheduled using static priorities ($t_1 > t_2 > t_3$) in a non-preemptive manner. Task t_1 is periodic with period a and a non-deterministic duration in $[10, b]$, where a and b are parameters. Task t_2 only has a minimal activation time of $2a$ and has a non-deterministic duration in $[18, 28]$ and finally t_3 is periodic with period $3a$ and a non-deterministic duration in $[20, 28]$. Each task is subject to a deadline equal to its period so that it must only have one instance active at all times. We ask for the parameter values that ensure that the system does not reach a deadline violation.[2] Algorithm IEFsynth produces the constraint $a \geq 34, b \geq 10, a - b \geq 24$ in 7.4 s on a Core i7/Linux computer, while algorithm RIEFsynth produces the constraint $a > \frac{562}{17}, b \geq 10, a - b > \frac{392}{17}$ in 12.7 s.

As illustrated here, the result of RIEFsynth is indeed a bit more precise (a graphic visualization is given in Figure 4.7: the improvement of RIEFsynth over IEFsynth is depicted in light green) but the main improvement is of course the guaranteed density of the result and the termination. Also, RIEFsynth is generally slower than IEFsynth and profiling shows that this is due to a decreased efficiency in computing the integer hull: we start each time from the whole symbolic state instead of starting from the successor of an already computed integer hull. This could be mitigated using a cache for the constraints generated in computing the integer hulls.

What's beyond...? In [ALR15], we also considered the case of unavoidability (AF). Contrarily to IEFsynth, the result output by IAFsynth (as defined in [JLR15]) when interpreted over *rational* valuations can even be wrong. Our extension RIAFsynth [ALR15] addresses these issues, and was implemented in Roméo. We also considered a similar extension of IM (not published in [ALR15] due to space constraints).

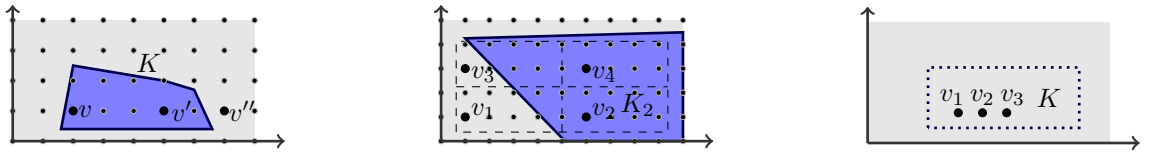
4.4 Towards distributed parameter synthesis

Parameter synthesis for PTA is often very expensive, often by an order of magnitude more than the non-parametric verification on TAs. This comes among other reasons from the fact that no efficient data structure is known for PTAs—in contrast to TAs which benefit from the relatively efficient DBMs.[3]

An option to partially leverage this problem is to use distributed verification over a *cluster*, i. e., a set of dozens, hundreds or sometimes thousands of computers (often called *nodes*), that have their own memory and processor, and communicate with each other over a network.

[2] The result is therefore the complement of the result given by IEFsynth and therefore an over-approximation containing no incorrect integer value.

[3] A parametric extension of DBMs ("PDBMs") was proposed in [Hun+02], but it benefits from two drawbacks: first, the successor of a PDBM is not unique, i. e., it may be a list of PDBMs, and second and most importantly, beyond the usual matrix, we still need to handle polyhedra (on parameters only), which still requires a polyhedra library.



(a) Graphical example (b) Redundancy (c) Choosing points

Figure 4.8 – Graphical representations and challenges

We show here that a parameter synthesis algorithm proposed during my PhD thesis, namely the *behavioral cartography*, can relatively be efficiently distributed over a cluster. We will then reuse the distributed algorithms in the subsequent sections (Section 4.5).

4.4.1 The behavioral cartography

The Inverse Method

Recall from Section 2.4 that the inverse method IM is a semi-algorithm solving the trace-preservation-synthesis problem. We call the result of IM a *tile*. Note that, in general, tiles have no predefined "shape": they are general polyhedra in $|P|$ dimensions that can have arbitrary size, number of vertices, and edge slope. The computation time of IM also greatly varies, from milliseconds to several hours, depending on the complexity of the model, and the size of the trace set.

The behavioral cartography

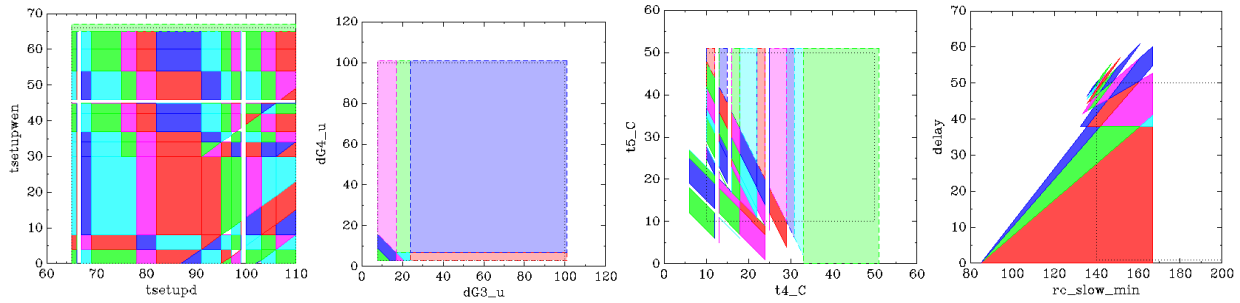
Given a bounded PTA $\mathcal{A}_{|bounds}$ the *behavioral cartography* (BC) [AF10] repeatedly calls IM on (some of the) integer points of $bounds$ (of which there is a finite number), so as to cover $bounds$ with tiles. The result gives a tiling of $bounds$ such that the trace set is the same for all valuations in a given tile.

Example 4.8. In Figure 4.8a, BC first considers point v, and computes $K = \mathsf{IM}(\mathcal{A}, v)$. Then, BC iterates on the subsequent points, all already covered by K, until it meets v'', that is not yet covered. Hence, BC will then compute $\mathsf{IM}(\mathcal{A}, v'')$, and so on, until all integer points in $bounds$ are covered.

BC can be used for several applications: first, it identifies the system robustness in the sense that, in each tile, parameters can vary as long as they remain in the tile, without impacting the system's discrete behavior. Second, BC can be used to perform parameter optimization; the weakest conditions of the input signal of an industrial asynchronous memory circuit (SPSMALL) were derived using BC [AS13]. Third, given a set of linear time properties (i. e., that can be verified on the trace set), it suffices to compute only once BC, and then to check each property on the trace set generated for each tile in order to know a complete (or nearly complete) set of parameter valuations satisfying each property.

Remark 4.1. BC does not guarantee the full, dense coverage of $bounds$ for two reasons.

1. IM may not terminate, as the corresponding problem is undecidable (Theorem 3.16).
2. Even if all calls to IM terminate, there is no (theoretical) guarantee than any point other than integers will be covered. IM may generalize integer points in the form of dense, rational-valued constraints, but it could happen that some (uncomputed) tiles do not contain any integer points. This sometimes happened in our experiments (e. g., in Figure 4.9a around $x = 100$ and $y = 55$).



(a) SPSMALL (b) Flip-flop circuit (c) Schedulability (d) RCP

Figure 4.9 – Examples of graphical behavioral cartographies in 2 dimensions

In practice, BC frequently covers (parts of) the parametric space beyond *bounds*; this is the case in Figures 4.9b to 4.9d (in Figure 4.9b, the entire parametric space is even covered).

4.4.2 Distribution policies

Our objective is to take advantage of the iterative nature of the cartography, and to distribute it on N processes. There is no theoretical obstacle in doing so, since all calls to IM are independent from each other. The challenge is rather to select efficiently the points on which IM is called, so that as few redundant constraints as possible are computed. The main goal will be to propose an efficient *distribution policy*,[4] i. e., a method to efficiently use the power of all nodes.

Static domain decomposition

A straightforward distribution policy is the static domain decomposition ("Static"). That is, the (hyper)rectangle *bounds* is split into N subdomains, and then each process is responsible for handling its own subdomain in an independent manner (with no communication).

For example, in Figure 4.8b, the domain *bounds* (the external dashed rectangle) is split into four equal subdomains (the four internal dashed rectangles); v_i, $1 \leq i \leq 4$ represents a possible first point on which to call IM in each subdomain. (K_2 in Figure 4.8b will be used later on.)

This static decomposition is straightforward but is not satisfactory for BC for three main reasons.

First, the general "shape" of the cartography is entirely arbitrary and unknown beforehand, since tiles can themselves have any shape. Figure 4.9 gives examples of cartographies in 2 parameter dimensions: although the geometrical distribution of the tiles of Figure 4.9a within *bounds* is rather homogeneous, this is not true at all for the others. For example, splitting the domain of Figure 4.9b (resp. Figure 4.9d) into four equal parts would be very unfair for the node responsible of the lower-left (resp. upper-right) subdomain, since most tiles are concentrated there; this would also be inefficient, since the other nodes will rapidly become idle.

Second, the geometrical distribution of the tiles says nothing on the *time* necessary to compute each tile. Even when the tiles are homogeneously located within *bounds*, some tiles may require much more time than others. Again, this would result in a loss of efficiency due to load unbalance since not all of the nodes are working actively.

Third, the absence of communication between nodes may result in redundant computations. Let us go back to the example of cartography in Figure 4.8b. Assume that node 2 finished first to compute a tile, say

[4] The proper word is a *scheduling* policy. However, since it (somehow) differs from the notion of scheduling used in Chapter 6, and in order to avoid any confusion, we use here distribution policy.

K_2. This tile not only covers the entire subdomain of node 2, leading to the termination of process 2, but it also covers node 4's subdomain entirely and a large part of node 2's subdomain. Without communication, these nodes will keep working without knowing that their subdomain has already been covered.

Defining more distribution policies

In [ACE14; ACN15], we proposed several distribution policies, that we evaluated. All of them rely on a "master-worker" scheme. In short, a node plays the role of a "master", that sends other nodes ("workers") some work to complete. Here, the work will typically consist of running an instance of IM on a given parameter valuation chosen by the master. When a worker finishes its computation, it sends back the result of IM and asks for another work—until all integer points in $bounds$ are covered by some tile. We review some of the distribution policies of [ACE14; ACN15] below. In all of them, the master also periodically informs the workers of the constraints computed by all nodes, so as to reduce the probability of redundant computations.

Sequential choice The first distribution policy (Seq) is a direct extension of the monolithic (i. e.,, non-distributed) algorithm: as in the non-distributed BC, the master enumerates all the points of $bounds$ in a sequential manner.

The main advantage of Seq is that it is inexpensive on the master's side. Its main drawback is the risk of redundant computations by the workers, due to the situation depicted graphically in Figure 4.8c: for instance, at the beginning, the N processes will ask for work, and the master will give them the first sequential N points, all very close to each other, with a high risk of redundant computation. (in Figure 4.8c, all three nodes would compute the same constraint K).

Random + sequential In the second distribution policy (Random) selects points randomly, and then in a second phase performs a sequential enumeration of all integer points in $bounds$ to check the full coverage of integers in $bounds$. This second phase is necessary to guarantee that all the integer points have been covered. The second phase starts after a given number of consecutive failed attempts to find an uncovered point randomly.

This policy is efficient to quickly cover a large part of the parameter domain, but becomes much slower when all points need to be enumerated in the second phase, especially for very large integer domains.

Shuffle The main problem of Random is the fact that the second phase, necessary to check the full coverage of integers, may be very costly. To alleviate this problem, we propose a new policy Shuffle that first computes *statically* a list of all integer points in $bounds$, then shuffles this list, and then selects the points of the shuffled list in a sequential manner. The sequential phase of Random is then dropped, at the cost of being able to compute, store statically and shuffle a potentially very large quantity of points.

Dynamic decomposition This policy consists in performing a dynamic decomposition of the parameter domain, and letting the workers selecting themselves the points, with some regulation by a coordinator. This policy does not entirely fit into the "master-worker" scheme, and is rather a "coordinator-worker" scheme.

Initially, the master splits in $bounds$ into N subdomains, and distributes the subdomains to the workers. In contrast to the previous policies, the workers are now responsible for checking whether all the points in their subdomain have been covered yet or not. This mechanism reduces the load on the master without leading to redundant point coverage checks. Then, when a worker has covered all the integer points in its subdomain (because the points are covered by tiles computed either by this worker, or by other workers),

Case study	Flip-flop4	RCP	Sched3-2	Sched3B-2	Sched3B-3	Sched5	SiMoP	Average
Model								
Clocks	5	6	13	13	13	21	8	
Parameters	4	2	2	2	3	2	2	
Integers in *bounds*	386,400	3,050	286	14,746	530,856	1,681	10,201	
Cartography								
# Tiles	190	19	59	71	378	177	48	
N_{max}	128	32	64	128	128	128	64	
Monolithic	1341.0	1992.0	46.0	61.2	865.0	3593.0	111.6	
Execution time at N_{max} (s)								
Static	33.0	2108.0	4.0	26.6	181.0	213.0	21.4	
Seq	2059.0	653.0	4.6	11.0	810.0	219.0	36.1	
Random	652.0	635.0	3.6	8.4	524.0	148.0	23.6	
Shuffle	670.0	624.0	3.1	7.6	243.0	140.0	18.7	
Subdomain	24.0	622.0	4.0	11.0	81.0	199.0	23.2	
Hybrid	24.0	624.0	3.1	7.6	81.0	140.0	18.7	
Ratio at N_{max} w.r.t. slowest at N_{max} (%)								
Static	2	100	56	100	22	78	59	60
Seq	100	31	64	41	100	80	100	74
Random	32	30	50	32	65	54	65	47
Shuffle	33	30	43	29	30	51	52	38
Subdomain	1	30	56	41	10	73	64	39
Hybrid	1	30	43	29	10	51	52	31
Speedup at N_{max} (%)								
Static	32	5	19	3	4	13	11	12
Seq	1	16	17	8	1	13	6	9
Random	2	17	22	10	1	19	10	11
Shuffle	2	17	25	11	3	20	12	13
Subdomain	44	17	19	8	8	14	10	17
Hybrid	44	17	25	11	8	20	12	20

Table 4.1 – Summary of distributed experiments

it informs the master; the master dynamically splits a subdomain (typically, one that has only been covered a little) and sends it back to the idle worker.

Experiments and conclusion We implemented these policies in a distributed version of IMITATOR, and we performed extensive experiments on two clusters of Grid'5000: Pastel (located in Toulouse, France), and Griffon (located in Nancy, France). Pastel is made of 140 nodes, each of which features two dual-core AMD Opteron 2218 running at 2.6 GHz, 8 GiB of RAM and a GigaEthernet interconnection network. Griffon is made of 92 nodes, each of which features two quad-core Intel Xeon L5420 running at 2.5 GHz, 16 GiB of RAM and both GigaEthernet and 20G InfiniBand network interconnection networks.

Among other metrics, we mainly evaluated the speedup, that evaluates the scalability of each algorithm: for each algorithm and each case study, we compute the time for this case study and this algorithm for N nodes divided by the time needed for a perfect algorithm (i. e., the monolithic time divided by N), and multiplied by 100. A number close to 100 means a very scalable algorithm, whereas a number close to 0 indicates an algorithm that does not scale well.

Our experiments are tabulated in Table 4.1 (a full version is available in [ACN15]). Overall, no algorithm is perfect on all case studies. The speedup also greatly varies depending on the case study. However, trends show that Static, Seq and Random overall scale badly with a speedup of 12, 9 and 11 respectively. In contrast, Shuffle (despite an average of 13) is often the best on the case studies with a small parameter domain (less than $100,000$ points). And Subdomain (with an average speedup of 17) behaves best on the larger case studies.

We therefore propose an hybrid policy (Hybrid), that selects Shuffle if the parameter domain is small, and Subdomain otherwise. This policy reaches a speedup of 20.

An average speedup of 20 % for Hybrid can seem relatively low; this means that a perfect distribution

algorithm (that would always divide the monolithic computation time by N) would be 5 times faster. Still, we find it promising. First, all distributed algorithms suffer from the time spent in communication, which always lowers the speedup. Second, this confirms that distributing BC is far from trivial, due to the unknown shape of the cartography, the unknown computation time for each tile, and the risk for redundant computations. Third, and most importantly, a speedup of 20 % means that, when using 128 nodes, the computation time is still divided by more than 25—which leads to an impressive decrease of the verification time.

4.5 EF-synthesis using reachability preservation

We address here a method to solve the EF-synthesis problem using an original manner. We assume a single "bad" location l_{bad}. Instead of attacking the problem by exploring (most of) the symbolic state space as in EFsynth, we will perform repeated computations of a limited part of the state space.

4.5.1 Reachability preservation

First, we introduce here an original procedure PRP$(\mathcal{A}, v)$, that synthesizes valuations *preserving the reachability* of l_{bad}.

Definition 4.5 (reachability preservation)**.** Given two TA $v(\mathcal{A})$ and $v'(\mathcal{A})$, we say that $v'(\mathcal{A})$ *preserves the reachability* of l_{bad} in $v(\mathcal{A})$ when l_{bad} is reachable in $v(\mathcal{A})$ if and only if l_{bad} is reachable in $v'(\mathcal{A})$.

PRP (standing for parametric reachability preservation) is at first close to a variant of IM (called IM^K, presented in [AS11] and briefly mentioned later in Section 5.1.1)), and then switches to an algorithm that resembles EFsynth:

- As long as no bad location is reached, PRP generalizes the trace set of $v(\mathcal{A})$ by removing v-incompatible states; this is done by negating v-incompatible inequalities, and returning the intersection of such negated inequalities, in the line of IM.
- When at least one bad location is met, PRP switches to an algorithm close to EFsynth, i. e., it simply gathers the constraints associated with the bad locations, and returns their union. However, a main difference with EFsynth is that PRP does not explore v-incompatible states: although this is not necessary to ensure correctness (in fact, this makes PRP not complete), this is a key heuristics to keep the state space of reasonable size.

We introduce PRP in Algorithm 5. It is a breadth-first exploration procedure that maintains the following variables: Passed (resp. Waiting) is the set of states computed at the previous (resp. current) iterations; Bad is a Boolean flag that remembers whether a bad location has been met; K_{good} is the intersection of the negation of all v-incompatible inequalities, that will be returned if no bad state is met; K_{bad} is the union of the projection onto P of all bad states, that will be returned otherwise; i remembers the exploration depth.

The procedure consists in a (potentially infinite) **while** loop. First, lines 4–6 take care of the v-incompatible states. These states are discarded from the exploration, i. e., they are removed from the set of waiting states (line 5). Then, if the exploration has not yet met any bad state, K_{good} is refined so as to prevent any such v-incompatible state (l, C) to be reached: the negation of $C{\downarrow_P}$ is added to K_{good}.

Second, lines 7–9 take care of the bad states. If any bad state is reached (line 7), then the Bad flag is set to true, the union of the projection onto P of the constraints associated with these bad states is added to K_{bad}, and these states are discarded, i. e., their successor states will not be computed (line 9).

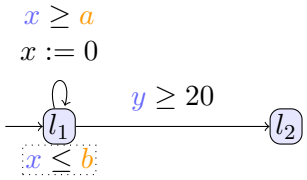


Figure 4.10 – An example of a PTA $\mathcal{A}_1$

The third part is a classical fixpoint condition: if no new state has been met at this iteration (line 10), then the result is returned, i. e., either K_{bad} if some bad states have been met, or K_{good} otherwise. If new states have been met, then the procedure explores one step further in depth (line 13).

Algorithm 5: PRP$(\mathcal{A}, v)$ [And+15]

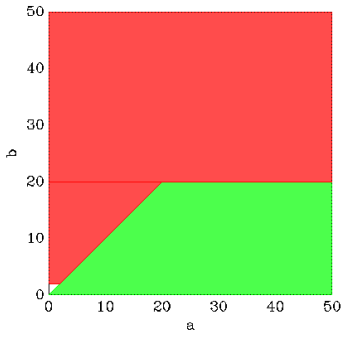
input : PTA $\mathcal{A}$ of initial state s_0, parameter valuation v
output: Constraint preserving the reachability of l_{bad} in $v(\mathcal{A})$

```
1  Passed ← ∅ ;  Waiting ← {s_0} ;
2  Bad ← false ;  K_good ← ⊤ ;  K_bad ← ⊥ ;  i ← 0
3  while true do
4      foreach v-incompatible state (l, C) in Waiting do
5          Waiting ← Waiting \ {(l, C)}
6          if Bad = false then K_good ← K_good ∧ ¬C↓_P ;
7      foreach bad state (l_bad, C) in Waiting do
8          Bad ← true ;  K_bad ← K_bad ∨ C↓_P ;
9          Waiting ← Waiting \ {(l_bad, C)}
10     if Waiting ⊆ Passed then
11         if Bad = true then return K_bad else return K_good ;
12     Passed ← Passed ∪ Waiting ;
13     Waiting ← Succ(Waiting) ;  i ← i + 1
```

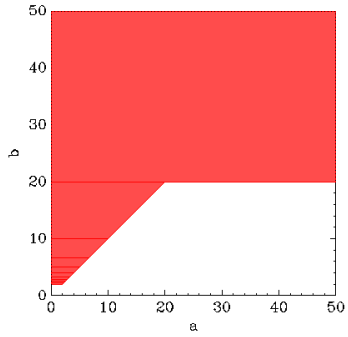
Theorem 4.3 ([And+15]). *Let $\mathcal{A}$ be a PTA, and v a parameter valuation. Suppose PRP$(\mathcal{A}, v)$ terminates with result K. Then, $v \models K$ and, for all $v' \models K$, l_{bad} is reachable in $v(\mathcal{A})$ iff l_{bad} is reachable in $v'(\mathcal{A})$.*

PRP may not terminate, which is natural since we showed that the reachability preservation emptiness (which is very similar to EF-emptiness) is undecidable [And+15]. Furthermore, even if it terminates, the result output by PRP may be non complete; in fact, this is *designed on purpose* (since we stop the exploration of v-incompatible states) so as to prevent a too large exploration. Enlarging the output constraint can be done by repeatedly calling PRP on other points than v, which will be done in the following.

Example 4.9. Consider the PTA $\mathcal{A}_1$ in Figure 4.10 [JLR15], with clocks x and y and parameters a and b. Assume $l_{bad} = l_2$. EFsynth$(\mathcal{A}_1, s_0, \{l_2\}, \emptyset)$ does not terminate, and neither does it if the range of the parameters is bounded from above (e. g., $a, b \in [0, 50]$).



(a) PRPC



(b) EFsynth

Figure 4.11 – EF-synthesis using PRPC and EFsynth for $\mathcal{A}_1$

Let us now apply PRP to $\mathcal{A}_1$. For valuation $v_1 : (a \leftarrow 20, b \leftarrow 10)$, PRP outputs $20 > b \wedge a > b \wedge b \geq 0$, which prevents the reachability of l_{bad}. For valuation $v_2 : (a \leftarrow 0, b \leftarrow 40)$, PRP does not terminate.

4.5.2 EF-synthesis

We saw in Section 4.4.1 that, given a bounded parameter domain, IM can be iterated on integer points to perform a behavioral cartography. In fact, PRP can be used in place of IM within BC, giving birth to a procedure PRPC (see Algorithm 6). PRP is called repeatedly with as an argument the first integer point not yet covered by any constraint (line 2 in Algorithm 6).

Algorithm 6: PRPC($\mathcal{A}$, *bounds*) [And+15]

input : PTA $\mathcal{A}$, bounded parameter domain *bounds*
output: Set K of constraints over the parameters (initially empty)

1 **while** *there are integer points in bounds not covered by* K **do**
2 Select an integer point v in *bounds* not covered by K
3 $K \leftarrow K \cup \mathsf{PRP}(\mathcal{A}, v)$
4 **return** K

In reality, PRPC maintains (and returns) *two* constraints, one for good valuations, and the other for bad valuations (not shown in Algorithm 6 as it would require to modify PRP itself to return a "flag" indicating whether the constraint is good or bad).

In the general case, PRPC may not terminate, due to the non-termination of PRP. However, it is possible to set up a maximum exploration depth for PRP: when this depth is reached, the algorithm stops. If some bad states have been met, the resulting constraint can be safely used (from a technical result in [And+15]); otherwise the constraint is just discarded and the reference point on which PRP was called will never be covered. In this case, termination of PRPC is always guaranteed, with a partial result (some integer points may still be uncovered).

Example 4.10. Consider again the PTA $\mathcal{A}_1$ in Figure 4.10, and let us apply EFsynth and PRPC with a bounded exploration depth of 10; this will result in correct under-approximations (from a technical result in [And+15]). We apply PRPC to an unconstrained model with *bounds* : $a, b \in [0, 50]$. We apply EFsynth to a model where a and b are constrained to be in $[0, 50]$. We give in a graphical manner in Figure 4.11a (resp. Figure 4.11b)

the results output by PRPC (resp. EFsynth). PRPC synthesizes all the good parameter valuations (below, in green), i. e., that do not reach l_2, and all the bad parameter valuations (above, in red), i. e., that reach l_2, with the exception of a small area near $(0, 0)$ (in white). All constraints output by PRPC are infinite (which is not shown in the figure), and hence cover the whole part outside *bounds* too. As of EFsynth, the same bad valuations as for PRPC are covered, but only within *bounds*, and no information is given about the good valuations. Hence, since EFsynth was stopped prematurely, no information can be given for the non-covered part: in particular, the white part of *bounds* cannot be decided, whereas PRPC covers everything except the small area near $(0, 0)$. This is a major advantage of PRPC over EFsynth in terms of precision of the result. Also recall that EFsynth covers only (a part of) *bounds* whereas PRPC covers here the whole parameter space beyond *bounds*.

Experiments We implemented PRP and PRPC in IMITATOR and conducted experiments in [And+15] to compare EFsynth, BC and PRPC.

PRPC dramatically outperforms BC for all case studies. This is due to the fact that the constraints output by PRP (that preserve only non-reachability) are much weaker than those output by IM (that preserve trace set equality). Second, PRPC compares rather well with EFsynth, and is faster on three case studies; PRPC furthermore outputs a more valuable constraint for $\mathcal{A}_1$ (see Example 4.10). PRPC can even verify case studies that EFsynth cannot.

Then, we can also use a distributed version of PRPC in the line of the distribution policies for BC in Section 4.4. We used the Subdomain as a distribution policy for PRPC, and used a cluster with 12 nodes. The distributed version of PRPC is shown to be faster than PRPC for all case studies. Most importantly, the distributed PRPC outperforms EFsynth for all but two case studies. The good timing efficiency of PRPC is somehow surprising, since it was devised to output a more precise result and to use less memory, but not necessarily to be faster. We believe that PRPC allows to explore small state spaces at a time and, despite the repeated executions, this is less costly than handling a large state space (as in EFsynth), especially when performing equality checks when a new state is computed.

4.6 Compositional synthesis for parametric event-recording automata

Here, I synthesize several lines of works, the ultimate goal of which is to perform compositional parameter synthesis. First, we introduce parametric event recording-automata (Section 4.6.1) as a subclass of PTAs. Then, we show propose an algorithm to learn an unknown (non-parametric) event-recording automaton by interacting with a teacher (Section 4.6.2), and we use this learning algorithm to perform compositional verification (Section 4.6.3). Finally, we define a new framework reusing both non-parametric compositional verification and the PRPC algorithm presented above, so as to perform compositional parameter synthesis (Section 4.6.4).

4.6.1 Parametric event-recording automata

Event-recording automata (ERAs) [AFH99] are a subclass of timed automata, where each action label is associated with a clock such that, for every edge with a label, the associated clock is reset. We propose here a parametric extension of ERAs, following the parameterization of TAs into PTAs.

Formally, let Σ be a set of actions: we denote by X_Σ the set of clocks associated with Σ, i. e., $\{x_\sigma \mid \sigma \in \Sigma\}$. A Σ-guard is a guard on $X_\Sigma \cup P$.

Definition 4.6 ([AL17a]). A *parametric event-recording automaton (PERA)* is a tuple $(\Sigma, L, l_0, F, P, I, E)$, where:

1. Σ is a finite set of actions,

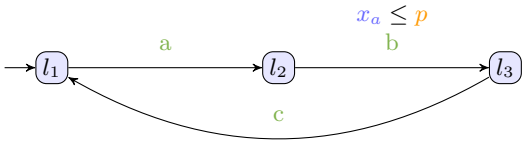


Figure 4.12 – An example of a PERA

2. L is a finite set of locations,
3. $l_0 \in L$ is the initial location,
4. $F \subseteq L$ is a set of accepting locations,
5. P is a finite set of parameters,
6. I is the invariant, assigning to every $l \in L$ a Σ-guard $I(l)$,
7. E is a finite set of edges $e = (l, g, \sigma, x_\sigma, l')$ where $l, l' \in L$ are the source and target locations, $\sigma \in \Sigma$, x_σ is is the clock to be reset, and g is a Σ-guard.

Just as for ERAs, PERAs can be seen as a syntactic subclass of PTAs: a PERA is a PTA for which there is a one-to-one matching between clocks and actions and such that, for each edge, the clock corresponding to the action is the only clock to be reset.

Following the conventions used for ERAs, we do not explicitly represent graphically the clock x_σ reset along an edge labeled with σ: this is implicit.

Example 4.11. Figure 4.12 depicts an example of PERA with 3 actions (and therefore 3 clocks x_a, x_b and x_c), and one parameter p. Only clock x_a is used in a guard.

Since ERAs are strictly less expressive than TAs w.r.t. the timed language, we may wonder whether some problems undecidable for PTAs become decidable for PERAs. In fact, since PERAs mostly impose restrictions on the timed language of a PTA, the proofs of undecidability that do not fundamentally rely on the language can be kept (with mild modifications). For example, our proof in Lemma 3.4 does not assume any action on the transitions; therefore, to obtain a PERA, when a clock x is reset, it suffices to label the transition with the action σ this clock is associated with. When no clock is reset, we can reset a fresh clock never used elsewhere in the PERA and label the transition with its associated action. Finally, when more than one clock is reset, a possibility is to duplicate the transition into several transitions in 0-time, each of them resetting exactly one clock (some care may be needed to avoid unwanted behaviors). In particular, the EF-emptiness problem remains undecidable for PERAs.

Theorem 4.4 ([AL17b]). *The EF-emptiness problem is undecidable for PERAs, even with bounded parameters.*

Proof idea. By adapting to PERAs the proof of Lemma 3.4. □

We conjecture that the proofs of undecidability of the emptiness of other quantifiers (AF, EG, AG) could be adapted in a similar fashion to PERAs.

However, a less trivial problem is whether the language- and trace-preservation-emptiness problems are decidable for PERAs. Indeed, both problems heavily rely on actions. In particular, the proof of [AM15] cannot be reused, as it fundamentally relies on the fact that all transitions should be labeled with the same action—which is not possible in a PERA, as it would imply that the same clock is reset on all transitions. Still, we proposed a new proof of undecidability for the language-preservation-problem.

Theorem 4.5 ([AL18]). *The language-preservation-emptiness problem is undecidable for PERAs.*

Proof idea. We use the original definition of the language of [AM15], i. e., the set of all maximal words (no accepting locations are used). We allow all possible infinite (untimed) words for the reference valuation. The main idea is as follows: for other valuations, we also allow all possible infinite words, and we allow entering a 2-counter machine encoding; if the machine does not halt, the encoding will block, therefore adding a new finite word which is not part of the language of the reference valuation. If the machine halts, thanks to a self-loop, the word simulating the machine is infinite, and therefore part of the language of the reference valuation. □

This proof can be adapted to other definitions of the languages, just as in Section 3.3.8. In addition, we conjecture it can be adapted for L/U-PERAs, following the reasoning used in Theorem 3.17.

However, the proof of Theorem 4.5 requires unbounded parameters, and the case of bounded PERAs is therefore open. In addition, the trace-preservation-emptiness seems to pose practical problems, and we were not able to solve it: it is open too.

The negative results in Theorems 4.4 and 4.5 rule out the possibility to perform exact synthesis for PERAs. Still, in the following, we will propose an approach for EF-synthesis that is sound, though maybe not complete: the synthesized valuations are correct, but some may be missing. More pragmatically, we aim at improving the synthesis efficiency.

4.6.2 Learning event-recording automata

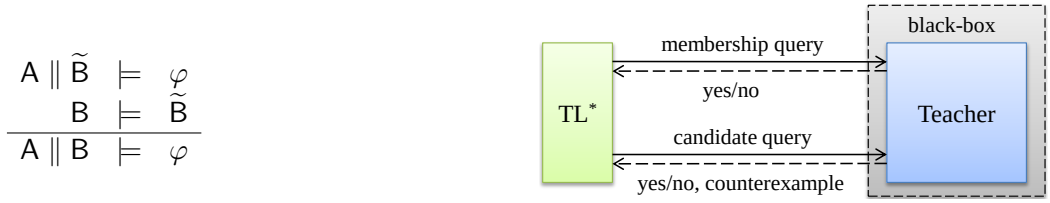
Learning an unknown system through a teacher (that has an access to the system) has been the object of several lines of works. In [Ang87], the L^* algorithm learns an unknown language described using a finite-state automaton using two types of queries to the teacher: first, a *membership query* asks whether a given word is accepted by the unknown automaton. After several membership queries, the L^* algorithm constructs a candidate automaton, and makes a *candidate query* for it. That is, it asks whether the automaton accepts the same language as the candidate automaton. The L^* algorithm is sound and complete.

Learning timed automata is not possible, because the language inclusion (required by the candidate query) is undecidable for TA [AD94]. The class of ERAs is however a good candidate to define a timed extension of the L^* algorithm, since the language inclusion becomes decidable [AFH99].

In [GJL10], L^* is extended to ERAs with three new algorithms TL^*_{sg}, TL^*_{nsg}, and TL^*_{s}. Their learning algorithms deal with *timed words*, which makes the complexity blow up as the learning algorithm has to actively infer the time condition of each event by performing successive guesses.

In [Lin+11], we proposed a new algorithm named TL* for learning ERAs. Different from [GJL10], it relies on *guarded words*, i. e., the counterexample returned by the teacher already contains some clocks and some clock guards, which makes the learning process easier. We proved that the learning process terminates in a finite number of iterations with a minimal number of locations in the learned ERA. In addition, our algorithm TL* works in polynomial time; in contrast, the algorithms of [GJL10] may yield an ERA that can be doubly exponentially larger than the ERA representing the same language to be learned, as the algorithms may have to explore the entire zone graph.

A main drawback of our algorithm TL* is that it relies on guarded words, which can be seen as a sort of "cheating": in a real-world application, the teacher may not want (or be able) to return a guarded



(a) AGR proof rule

(b) TL* and Teacher

Figure 4.13 – AGR proof rule (left) and TL* (right)

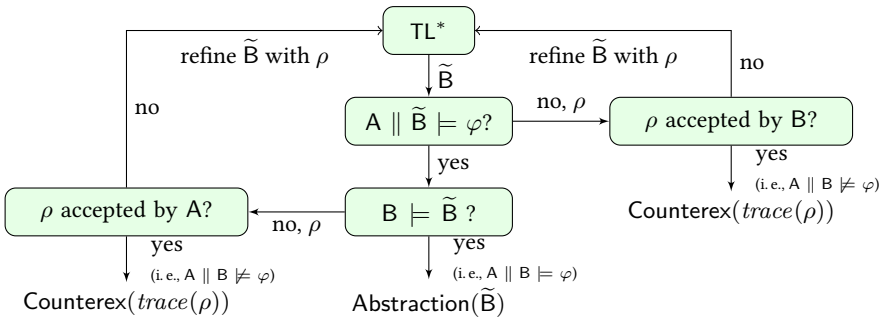


Figure 4.14 – LearnAbstr(B, A, φ)

word, but rather a timed word. However, this setting is very useful in the framework of compositional verification—which is the purpose of the next section.

4.6.3 Compositional verification of event-recording automata

Figure 4.13a recalls the common proof rule used in Assume-Guarantee Reasoning (AGR), which is one of the compositional verification techniques. Given two components A, B and a safety property φ, the proof rule tells that if A can satisfy the property φ under an assumption $\widetilde{B}$ and B can guarantee this assumption $\widetilde{B}$, then we can conclude that $A \parallel B$ satisfies φ. One can observe that the proof rule decomposes one model checking problem ($A \parallel B \models \varphi$) into two sub problems ($A \parallel \widetilde{B} \models \varphi$ and $B \models \widetilde{B}$).

A main issue in AGR is that computing the abstraction $\widetilde{B}$ requires non-trivial human creativity. A way to compute $\widetilde{B}$ automatically is to use learning, i. e., the TL* algorithm presented above. In [Lin+14], we proposed an automated procedure for compositional verification of ERAs. Figure 4.14 shows our overall procedure LearnAbstr(B, A, φ) that returns either an assumption (denoted by Abstraction($\widetilde{B}$)) when it is proved that $A \parallel B \models \varphi$ holds, or a counterexample (denoted by Counterex(τ)) otherwise. The two condition checks in Figure 4.14 ($A \parallel \widetilde{B} \models \varphi$ and $B \models \widetilde{B}$) can be done by model checking, and counterexamples given by model checking can also serve as counterexamples to the TL* algorithm. Counterex and Abstraction are "tags" containing a value. We omit the technical details of LearnAbstr(B, A, φ) here. Interested readers are referred to [Lin+14].

This framework can be extended to the case of more than two components.

Experimental validation Lin Shang-Wei implemented this algorithm into a tool [Lin+12], reusing in part the PAT model-checking library [Sun+09]. Experiments on a set of *ad-hoc* case studies showed a dramatic decrease of the computation time when compared to the monolithic (non-compositional) verification. The efficiency is sensitive to the partitioning of the components, and we proposed heuristics. Uppaal runs out of memory on most of these case studies (converted into TAs), while our algorithm terminates in a few seconds with a very limited memory usage (see [Lin+14] for details).

Algorithm 7: CompSynth(A, B, $bounds, T$)

input : PERA A, ERA B, parameter domain $bounds$, subset T of locations
output: Good and bad constraint over the parameters

1 $K_{bad} \leftarrow \bot$; $K_{good} \leftarrow \bot$
2 **while** $bounds \cap \mathbb{N} \cap (K_{bad} \cup K_{good}) \neq \emptyset$ **do**
3 Pick v in $bounds \cap \mathbb{N} \cap (K_{bad} \cup K_{good})$
4 **switch** *LearnAbstr*(B, v(A), *AG*¬T) **do**
5 **case** Abstraction($\widetilde{\mathsf{B}}$) **do**
6 $K_{good} \leftarrow K_{good} \cup$ PRP(A $\parallel \widetilde{\mathsf{B}}, v, T$)
7 **case** Counterex(τ) **do**
8 $K_{bad} \leftarrow K_{bad} \cup$ ReplayTrace(A $\parallel$ B, τ)
9 **return** (K_{good}, K_{bad})

4.6.4 Compositional parameter synthesis

Combining our framework for (non-parametric) compositional verification together with the reachability-preservation synthesis algorithm PRPC, we can define a framework for compositional parameter synthesis [AL17b]. The key idea is to partition parametric components in A and non-parametric components in B, and try to learn an abstraction of B, using a valuated version of A (i. e., v(A)) in the AGR framework of Figure 4.13a. Then, we iterate over integer-valuations of a bounded domain as in PRPC.

Our procedure (given in Algorithm 7) takes as arguments a set of PERA components A, a set of ERA components B, a bounded parameter domain $bounds$ and a set of locations to be avoided. We maintain a safe non-convex parameter constraint K_{good} and an unsafe non-convex parameter constraint K_{bad}. Then CompSynth iterates on integer points: while not all integer points in $bounds$ are covered, such an uncovered point v is picked (line 3). Then, we try to learn an abstraction of B w.r.t. v(A) (line 5) so that T is unreachable. If an abstraction is successfully learned, then PRP (given in Algorithm 5) is called on v and the abstract model A $\parallel \widetilde{\mathsf{B}}$ (line 6); the constraint K_{good} is then refined. Note that K_{good} is refined because, if an abstraction is computed, then necessarily the property is satisfied and therefore the (abstract) system is safe. Alternatively, if LearnAbstr fails to compute a valid abstraction, then a counterexample trace τ is returned (line 7); then this trace is "replayed" using a procedure ReplayTrace (line 8), that synthesizes exactly the parameter valuations corresponding to a (necessarily finite) trace [AL17b], and the constraint K_{bad} is updated.

Experiments We implemented our method in a toolkit made of IMITATOR [And+12] and of CV (Compositional Verifier), a new implementation (in C++) of the proposed learning-based compositional verification framework for ERAs initially presented in [Lin+12]. The leading tool is IMITATOR, that takes the input model (in the IMITATOR input format), and eventually outputs the result. IMITATOR implements both algorithms CompSynth and ReplayTrace, while CV implements LearnAbstr. The interface between both tools is handled by a Python script, that is responsible for retrieving the abstraction of B computed by CV and re-parameterizing the components A.

Experiments are tabulated in Table 4.2 (full details are available in [AL17b]). CompSynth is faster than EFsynth for most case studies. In addition, whereas EFsynth often does not terminate, CompSynth always outputs a result (except for one case study: the Fischer mutual exclusion protocol). In some cases, EFsynth is much faster because it immediately derives $\bot$, whereas CompSynth has to compute the abstraction first. Even in these unfavorable cases, CompSynth is never much behind EFsynth. This suggests that CompSynth

Case study	$\#\mathcal{A}$	$\#X$	$\#P$	Spec	EFsynth	PRPC		CompSynth			
						#iter	total	#abs	#c.-ex.	learning	total
FMS-1	6	18	2	1	0.299	2	0.654	1	1	0.074	**0.136**
				2	**0.010**	1	0.372	0	1	0.038	0.046
				3	0.282	1	0.309	1	0	0.090	**0.242**
FMS-2	11	37	2	1	T.O.	-	T.O.	1	1	84.2	**88.9**
				2	T.O.	-	T.O.	1	0	81.4	**85.2**
				3	**0.051**	-	T.O.	0	2	1.10	2.44
				4	**0.062**	-	T.O.	0	1	1.42	1.53
				5	T.O.	-	T.O.	1	0	31.4	**40.8**
				6	T.O.	-	T.O.	1	0	37.2	**42.4**
AIP	11	46	2	1	0.551	-	T.O.	0	1	0.086	**0.114**
				2	2.11	-	T.O.	0	1	1.22	**1.25**
				3	**3.91**	-	T.O.	0	1	8.50	8.54
				4	**0.235**	-	T.O.	1	1	8.39	8.42
				5	T.O.	-	T.O.	1	0	0.394	**0.871**
				6	T.O.	-	T.O.	1	0	5.32	**9.58**
				7	T.O.	-	T.O.	1	0	1.76	**3.19**
				8	T.O.	-	T.O.	1	0	1.13	**4.35**
				9	T.O.	-	T.O.	1	1	0.762	**1.84**
				10	**0.022**	-	T.O.	0	1	0.072	0.094
Fischer-3	5	12	2		**2.76**	4	14.0	0	1	-	T.O.
Fischer-4	6	16	2		T.O.	-	T.O.	0	1	-	T.O.

Table 4.2 – Summary of compositional verification experiments

may be preferred to EFsynth for PERAs benchmarks.

Interestingly, in almost all benchmarks, at most one abstraction (for good valuations) and one counter-example (for bad valuations) is necessary for CompSynth. In addition, most of the computation time of CompSynth (71 % in average) comes from LearnAbstr; this suggests to concentrate our future optimization efforts on this part. Perhaps an on-the-fly composition mixed with synthesis could help.

For Fischer, CompSynth is very inefficient: this comes from the fact that the model is strongly synchronized, and the abstraction computation does not terminate within 600 s. In fact, in both cases, LearnAbstr successfully derives very quickly a counter-example that is used by CompSynth to immediately synthesize all "bad" valuations; but then, as LearnAbstr fails in computing an abstraction, the good valuations are not synthesized. Improving the learning phase for strongly synchronized models is on our agenda.

What's beyond...? Of course, CompSynth is close to PRPC and could therefore benefit from a distributed version, following the distributed policies presented in Section 4.4.

4.7 Perspectives

Clock elimination Our clock elimination technique for PTAs is fairly simple, but could be improved with the detection of *quasi-equal* clocks: this line of work [Her+12; MWP13; HW16] consists in detecting clocks not necessarily equal in all locations, but that may differ only in 0-time before their reset (e. g., in different locations met in 0-time that reset successively these clocks). An additional difficulty is that the notion of quasi-equality may depend on the parameter valuations, since it may have an impact on the "0-time".

Distributed verification Beyond, our distributed procedure in [ACE14; ACN15], I am interested in *swarm* parameter synthesis, i. e., with many autonomous machines. Several procedures for swarm verification using Uppaal were proposed in [ZNL16b]; adapting these procedures to synthesis and developing new ones would be of interest.

Compositional verification When considering compositional parameter synthesis, our algorithm in [AL17b] may not terminate; therefore, an interesting future work would be to address the full class of parametric timed automata. Indeed, although language inclusion is undecidable for timed automata, an efficient algorithm was proposed in [Wan+14], that often terminates. Combining this with our synthesis framework [AL17b] would allow us to address the full class of PTAs (without guarantee on termination). Also, combining the compositional verification mechanism of [Aşt+16] (that uses completely different techniques from ours) would be an interesting future work.

Machine learning The paradigm of machine learning could help to increase the efficiency of parameter synthesis. We studied preliminary results in [Li+17], where we used classification techniques to "guess" potential parameter constraints after repeated calls to the non-parametric model checker Uppaal. Only after a constraint is guessed, IMITATOR is invoked to verify this constraint, leading to a dramatic gain in computation time when compared to a purely parametric analysis. So far, only "simple" constraints can be efficiently guessed; this approach should therefore be extended to more complex constraints. More generally, combining testing (or verification of a non-parametric timed system) with parameter synthesis looks promising, so as to gain from the efficiency of the former.

Stochastic optimization Stochastic optimization, e. g., used to falsify properties of hybrid systems in [AH15], seems also an interesting paradigm to exhibit synthesize optimal (or near-to-optimal) parameter valuations w.r.t. to an optimization criterion.

Chapter 5

Synthesis algorithms

After studying decidability (Chapter 3) and proposing techniques to speedup the verification (Chapter 4), we introduce here synthesis (semi-)algorithms to solve practical problems. We do not address the termination of these algorithms as a main goal, as underlying decision problems are almost always undecidable; however, whenever possible, we try to *improve* termination, i. e., make them terminate for larger classes of models.

These works fit into a line of works related to synthesis for parametric timed automata or parametric extensions of timed automata, regardless of their decidability (e. g., [Tra12; San15]). Finally, while looking for decidable subclasses, [CPR08; JLR15] also define semi-algorithms for (general) PTAs.

We first show that parameter synthesis using parametric timed automata and parametric time Petri nets can be used to measure the system robustness, in the sense of the measure of the admissible variability of the timing constants while preserving the untimed language (Section 5.1).

Second, we introduce a synthesis algorithm for parametric model checking under the non-Zeno assumption (Section 5.2).

We also propose synthesis algorithms in the setting of two richer models than parametric timed automata, i. e., augmented with controllable actions seen as parameters (Section 5.3) and with interval probabilistic distributions (Section 5.4).

5.1 Parameter synthesis and robustness

5.1.1 Varying the definition of robustness

The inverse method [And+09b; AS13] (recalled in Section 2.4) is a semi-algorithm solving the trace-preservation-synthesis problem. In [AS11], we relaxed the definition of the preservation of the untimed language and designed variants of IM. Given a PTA $\mathcal{A}$ and a reference parameter valuation v, the constraint synthesized by these variants preserve:

- the reachability of locations of $v(\mathcal{A})$: algorithm $\mathsf{IM}_{\subseteq}$;
- the fact that each trace is a prefix of a trace of $v(\mathcal{A})$: algorithm IM^{K};
- the inclusion into the trace set of $v(\mathcal{A})$ and the preservation of at least one maximal trace: $\mathsf{IM}^{\cup}$.

In addition, all variants preserve safety properties: a location unreachable in $v(\mathcal{A})$ remains unreachable for any valuation satisfying the constraint synthesized by any of these algorithms.

We summarize in Table 5.1 the properties satisfied by these algorithms.

Property	IM	$\mathsf{IM}_{\subseteq}$	$\mathsf{IM}^{\cup}$	IM^{K}
Equality with the trace set of $v(\mathcal{A})$	√	×	×	×
Inclusion into the trace set of $v(\mathcal{A})$	√	×	√	√
Preservation of at least one trace of $v(\mathcal{A})$	√	×	√	×
Equality of location sets of $v(\mathcal{A})$	√	√	×	×
Preservation of non-reachability	√	√	√	√

Table 5.1 – Comparison of the properties of the variants of IM

Remark 5.1. The names of the algorithms may be a bit misleading (for example, IM^{K} could have been named $\mathsf{IM}_{\subseteq}$ as it guarantees a trace set included in that of $v(\mathcal{A})$); these names come from the operators used in the actual algorithm (for example, $\mathsf{IM}_{\subseteq}$ uses an inclusion test instead of an equality test as in IM; IM^{K} returns a constraint named "K" instead of the intersection of all constraints associated to the explored states; and so on). We nevertheless keep these names in this manuscript by consistency with [AS11; ACR17].

In addition, we showed that all these algorithms enhance the termination of IM. That is, given $\mathcal{A}$ and v, if IM terminates, then all variants terminate—but the converse is not true.

Finally, and this is probably the most interesting in practice, the constraint output by these variants is larger (i. e., contains more valuations) than that returned by IM.

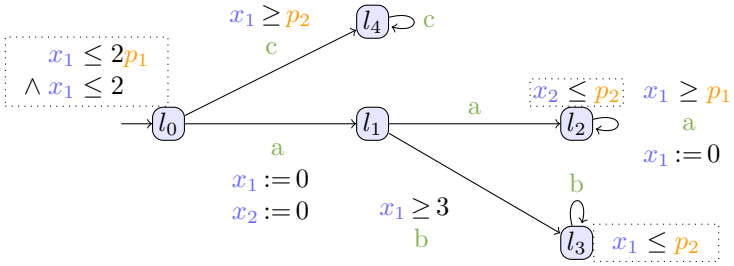


Figure 5.1 – A PTA $\mathcal{A}_{var}$ for comparing the variants of IM

Example 5.1. Let us consider the PTA $\mathcal{A}_{var}$ depicted in Figure 5.1. We consider the following v: $p_1 = 1 \wedge p_2 = 4$. In $v(\mathcal{A})$, location l_4 is not reachable, and can be considered as a "bad" location.

Let us suppose that a bad behavior of $\mathcal{A}_{var}$ corresponds to the fact that a trace goes into location l_4. Under v, the system has a good behavior. As a consequence, since all algorithms preserve the safety, the constraint synthesized by any algorithm also ensures l_4 remains unreachable.

We give in Figure 5.2 the four constraints synthesized by IM, $\mathsf{IM}^{\cup}$, IM^{K} and $\mathsf{IM}_{\subseteq}$, respectively. For each graphics, we depict in dark blue the parameter domain covered by the constraint, and in light red the parameter domain corresponding to a bad behavior. The "good" zone not covered by the constraint is depicted in very light gray. The dot represents v.

Implementation I implemented all these algorithms in IMITATOR, and experiments on a set of benchmarks show the practical interest of the variants in terms of memory and time, when compared to IM (see [AS11]).

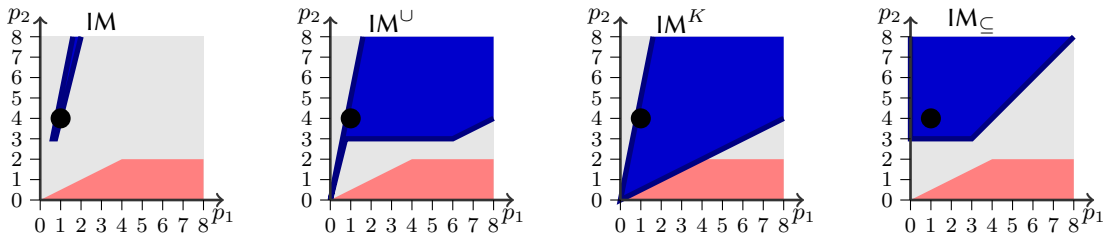


Figure 5.2 – Comparison of the constraints synthesized for $\mathcal{A}_{var}$

What's beyond...? We also considered combinations of the properties of these algorithms, such as $\mathsf{IM}^{K}_{\subseteq}$ and $\mathsf{IM}^{\cup}_{\subseteq}$. See [AS11] for details.

5.1.2 Precise robustness in time Petri nets

In [APP13], we redefined the inverse method in the context of parametric time Petri nets (PTPNs), and showed how it can be used to measure the robustness. Let us first recall PTPNs, as they will be used here and in Section 5.1.3 below.

(Parametric) time Petri nets

Parametric time Petri nets (PTPNs) [TLR09] are a parametric extension of time Petri nets [Mer74], where the temporal bound of each transition can either be a rational number, ∞ or a parameter.

Definition 5.1 (parametric time Petri net)**.** A parametric time Petri net (PTPN) is a tuple $\mathcal{N} = (\mathcal{P}, \mathcal{T}, P, pre, post, pefd, plfd, M_0, K_0)$ where

- $\mathcal{P}$ and $\mathcal{T}$ are non-empty, disjoint sets of *places* and *transitions* respectively,
- $P = \{p_1, \ldots, p_M\}$ is a finite set of *parameters*,
- pre and $post$ map each transition $t \in \mathcal{T}$ to its (nonempty) *preset*, denoted by ${}^\bullet t = pre(t) \subseteq \mathcal{P}$, and its (possibly empty) *postset*, denoted by $t^\bullet = post(t) \subseteq \mathcal{P}$;
- functions $pefd : \mathcal{T} \to \mathbb{Q}_+ \cup P$ and $plfd : \mathcal{T} \to \mathbb{Q}_+ \cup P \cup \{\infty\}$ and associate the *earliest firing delay* $pefd(t)$ and *latest firing delay* $plfd(t)$ with each transition t,
- $M_0 \subseteq \mathcal{P}$ is the *initial marking*, and
- K_0 is the *initial constraint* over P giving the initial domain of the parameters, and must at least specify that the firing intervals are nonempty ($\bigwedge_{t \in \mathcal{T}} pefd(t) \leq plfd(t)$).

As usual, we graphically represent places as circles and transitions as rectangles. We write the time interval $[pefd(t), plfd(t)]$ next to the transition.

Example 5.2. Consider the PTPN in Figure 5.3. Initially, one token is in place pl_1 and one in pl_3 while other places contain no token. Transition t_1 can fire at least a_1 and at most b_1 time units after its input place (here only pl_1) contains a token; as this is the case, it will fire at least a_1 and at most b_1 time units after the system

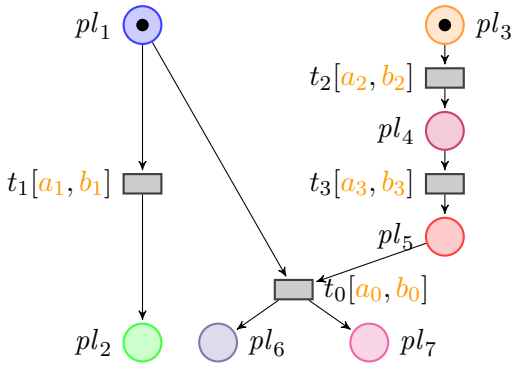


Figure 5.3 – An example of a PTPN

start—unless another transition (typically t_0) consumes the token of pl_1 before.

Given a parameter valuation v, we denote by $v(\mathcal{N})$ the non-parametric PTPN where all occurrences of a parameter p_i have been replaced by $v(p_i)$. We say that $v(\mathcal{N})$ is a *time Petri net* (TPN).

We consider only *safe* time Petri nets (TPNs), i. e., TPNs where there is never more than one token in a place.

We do not recall the semantics of time Petri nets, as it is available in many documents (e. g., [ACR17] using almost the same notations as in this manuscript).

A timed word is a possibly infinite sequence of pairs made of a transition and a *firing time*. A *sequence* associated to a timed word is the sequence of transitions, i. e., the untimed support of the timed word; it is the equivalent of a *trace* for PTAs. Given a TPN N, we denote by $Sequences(N)$ the set of sequences associated with all timed words of N, among which we distinguish the set $MaxSequences(N)$ of *maximal* sequences, i. e., sequences which are not the prefix of any other sequence.

Example 5.3. Consider the simple TPN in Figure 5.4a (from [Aks+16]). According to the semantics of TPNs, place C is unreachable, that is, there exists no reachable marking such that the number of tokens in C is greater than 0. Indeed, starting from marking A (i. e., a marking with 1 token in place A), t_1 can fire anytime between 1 and 2 time units after the system start. At time 2, t_1 must fire if it has not yet fired, because its associated interval is about to expire and no other transition is firable (t_2 will be firable right after time 2). Hence, C is unreachable.

The set of sequences of this TPN is therefore simply $\{(t_1)\}$, and its set of maximal sequences is the same.

Example 5.4. Consider again the PTPN in Figure 5.3. Fix $a_0 = b_0 = a_1 = b_1 = a_2 = b_2 = a_3 = b_3 = 1$. Then the set of maximal sequences of this TPN is $\{(t_1, t_2, t_3), (t_2, t_1, t_3)\}$.

Now fix $a_0 = b_0 = a_2 = b_2 = a_3 = b_3 = 1$, $a_1 = 1$ and $b_1 = 4$. The set of maximal sequences of this TPN becomes $\{(t_1, t_2, t_3), (t_2, t_1, t_3), (t_2, t_3, t_0)\}$.

PTPNs can be given a symbolic semantics (initially defined in [TLR09], and recalled using our notations in [ACR17]), that resembles very much the symbolic semantics of PTAs (Definition 2.6).

Precise robustness

In this section, just as in [APP13], we will consider an extension of PTPNs that allow *inhibitor arcs* [TLR09], abbreviated as (P)ITPNs. Inhibitor arcs can stop time elapsing, and can be seen as equivalent to stopwatches

(a) Example of undesired reachability

(b) Example of unlikely reachability

Figure 5.4 – Examples of non-robust (I)TPNs

in timed automata [CL00]. Inhibitor arcs do have an impact on the decidability of non-parametric time Petri nets [Rou04] but, in the parametric settings, as the interesting problems are undecidable, this does not fundamentally change the results.

Example 5.5. In Figure 5.4b, the arc connecting place D to transition t_3 is inhibiting D: that is, if D contains a token, then t_3 cannot fire.

Motivating the robustness Consider again the PTPN in Figure 5.4a. We showed in Example 5.3 that place C cannot be marked. Now suppose that the upper bound of the firing interval of t_1 is increased, even by an infinitesimal duration. Then, t_2 is firable immediately after time 2, and C can be reached in some executions.

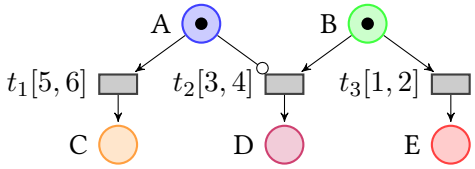
Now consider the ITPN in Figure 5.4b. According to the semantics of ITPNs, place E is reachable. Indeed, starting from a marking AB (i. e., a marking with 1 token in place A and 1 token in place B), t_1 can fire at time 1, giving marking CB. Then, after a null duration, t_3 can fire due to the absence of token in D. Finally, again after a null duration, t_2 finally fires. This sequence of transitions is unlikely to happen in practice due to delays exactly equal to zero; if the bounds of t_1 or the lower bound of t_3 become slightly larger, or the bounds of t_2 becomes slightly smaller, E becomes unreachable.

That is, both examples in Figures 5.4a and 5.4b are not robust w.r.t. the sets of sequences, as an infinitesimal change of the bounds may yield different sets of sequences.

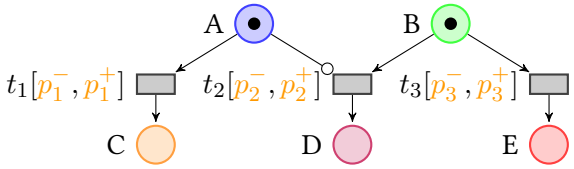
Precise robustness In [APP13], we proposed a precise version of robustness—precise because it is multidimensional, i. e., we measure the variability of all timing bounds independently. Our method is only a semi-algorithm and therefore may not terminate. We used a version of the inverse method IM adapted to parametric time Petri nets; despite some technical specificities for PTPNs, it is essentially the same algorithm as for PTAs (see Section 2.4) and we do not recall it (see [APP13]).

Theorem 5.1 ([APP13]). *Suppose IM$(\mathcal{N}, v)$ terminates with result K.*
Then, for all v', $Sequences(v(\mathcal{N})) = Sequences(v'(\mathcal{N}))$ iff $v' \models K$.

Now, assume an ITPN N. Assume a parameterized version $\mathcal{N}$ of N where each lower (resp. upper) bound of a transition t_i is replaced with a fresh parameter p_i^- (resp. p_i^+).



(a) An ITPN



(b) Parameterized version

Figure 5.5 – An ITPN and its parameterized version

Example 5.6. Consider the ITPN in Figure 5.5a. Its parameterized version is shown in Figure 5.5b.

Definition 5.2. An ITPN N is *robust with respect to linear-time properties* (or *LT-robust*) if there exists $\gamma > 0$ such that for any linear time property φ, $N' \models \varphi$ if and only if $N \models \varphi$, where N' is an ITPN similar to N where each timing bound c can be replaced with any value within $[c - \gamma, c + \gamma]$.

For example, the ITPN in Figure 5.5b is LT-robust (with e. g.,$\gamma = 1$), whereas the ITPNs in Figure 5.4 are not.

We showed in [APP13] that we can use the result of IM to measure the system robustness, i. e., to determine whether an ITPN is LT-robust. In addition, when it is not, we can exhibit which timing constants are i. e., critical, i. e., responsible for the non-robustness. Finally, we give a sufficient condition to improve the system robustness, i. e., to refine some values of the critical timing bounds so that the system becomes LT-robust.

This work can be put into perspective with the literature on TAs and TPNs: the simpler problem of robust untimed language preservation is already undecidable even for bounded TPNs [Aks+16]. In contrast, *language robustness* is decidable for TAs with some assumptions [San11]. However, in our case, we address the general class of (I)TPNs and, when our semi-algorithm terminates, we can evaluate the system robustness even if only some firing times can be enlarged or shrinked, while most of the literature considers that a system with at least one guard non-enlargeable (or non-shrinkable) is simply non-robust.

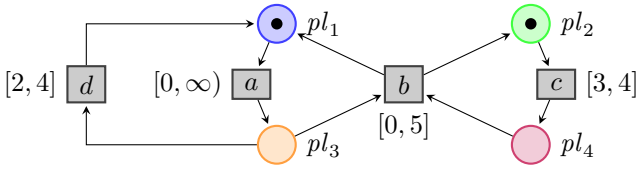
5.1.3 Robustness and partial orders

In [ACR17], we considered a concurrent setting, and showed that it is possibly to significantly relax the result of the variant of the inverse method IM^K when preserving the traces *up to partial orders*.

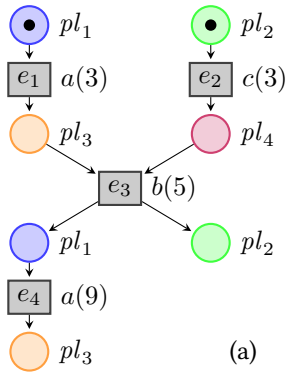
Due to the fact that in a TPN a given maximal sequence for a given valuation is also maximal for any other parameter valuation where this sequence is allowed (which is not the case in PTAs, notably due to invariants), the correctness of IM^K slightly differs from the PTA setting:

Theorem 5.2 ([ACR17]). *Suppose $IM^K(\mathcal{N}, v)$ terminates with result K. Then, for all v', $Sequences(v'(\mathcal{N})) \subseteq Sequences(v(\mathcal{N}))$ iff $v' \models K$.*

The result of IM^K can be seen as too rigid. Consider again the PTPN of Figure 5.3. Consider v_0 such that $a_0 = 0, b_0 = 3, a_1 = 0, b_1 = 1, a_2 = 2, b_2 = 3, a_3 = 1, b_3 = 2$. Because the initial parameter valuation v is such that $b_1 < a_2$, the constraint output by IM^K forces this ordering and allows only valuations for which the only maximal sequence possible is (t_1, t_2, t_3), like in $v(\mathcal{N})$.



(a) A TPN



(b) Graphical representation of a process

Figure 5.6 – TPN and process

With other parameter valuations, three other maximal sequences appear, viz., (t_2, t_1, t_3), (t_2, t_3, t_1), (t_2, t_3, t_1) and (t_2, t_3, t_0) (all already considered in Example 5.4). It is reasonable that a parameter synthesis method prevents valuations of the parameters which allow the last sequence, because it fires t_0 which differs qualitatively from the reference behavior. But the other sequences do not fire any undesired transition; they just reorder the firing of t_1, t_2 and t_3. Observing carefully the model, one even remarks that t_1 is actually *concurrent* to t_2 and t_3, and that the sequences (t_2, t_1, t_3) and (t_2, t_3, t_1) are simply obtained by changing the index where t_1 is inserted in the sequence (t_2, t_3).

A *process* is a representation of an execution of a (time) Petri net. Executed actions (called events) are not totally ordered, as in timed words. For untimed Petri nets, only causality orders the events. For time Petri nets, the firing time of each event can still be represented together with the event, but the partial-order causality indicates the structural dependencies between events due to creation and consumption of tokens.

An execution of a TPN N is represented as a labeled acyclic Petri net where every transition (called *event* and labeled by a transition t of N and a firing date) stands for an occurrence of t, and every place (called *condition* and labeled by a place p of N) refers to a token produced by an event in place p or to a token of the initial marking. The arcs represent the creation and consumption of tokens. Because fresh conditions are created for the tokens created by each event, every condition has either no input arc (if it is an initial condition) or a single input arc, coming from the event that created the token. Symmetrically, each place has no more than one output arc since a token can be consumed by only one event in an execution.

Example 5.7. Figure 5.6b shows an example process of the TPN in Figure 5.6a. This process corresponds to the sequential execution $\big((a,3),(c,3),(b,5),(a,9)\big)$. The dates of the events are in parentheses. Observe that this process also represents the timed word $\big((c,3),(a,3),(b,5),(a,9)\big)$.

An *abstract process* is a set of events of a TPN that represent an abstract representation of processes. In short (see [ACR17] for a complete definition), this set must be causally closed (the past of this set must be the set itself), and conflict free (the predecessors of any two events must be disjoint). Let $MaxProcesses(N)$ denote the set of maximal abstract processes of N.

Example 5.8. The PTPN of Figure 5.3 has two maximal abstract processes: one where transitions t_1, t_2 and

t_3 fire (giving rise to, resp. events e_1, e_2, e_3), the second with t_2, t_3 and t_0 (giving rise to, resp. events e_2, e_3 and e_0).

We now define a procedure IM^{K}PO (standing for "inverse method based on partial-orders") for synthesizing parameters in a PTPN $\mathcal{N}$ that guarantee the preservation of partial-order semantics. More precisely, given $\mathcal{N}$ and v, IM^{K}PO will synthesize valuations guaranteeing that the set of maximal processes of $v(\mathcal{N})$ contains the set of maximal processes of $v'(\mathcal{N})$ for any v' satisfying the constraint. Note that this requirement concerns only *maximal* processes: asking for preservation of all processes would limit the freedom in the interleavings of concurrent transitions. For the PTPN of Figure 5.3, the only (maximal) sequence feasible with the initial valuation v_0 (given above) is (t_1, t_2, t_3). Consider another valuation v that would force (t_2, t_1, t_3) (which we consider correct). A (non-maximal) timed word with only t_2 yields a (non-maximal) abstract process which is *not* feasible under v_0. On the other hand, the *maximal* abstract processes are the same for both valuations.

IM^{K}PO terminates for PTPNs where all the abstract processes are finite. It relies on the computation of the *unfolding* of the untimed support of the PTPN. Efficient tools exist for computing unfoldings [Kho12; Sch14]. The procedure IM^{K}PO$(\mathcal{N}, v_0)$ operates as follows:

1. Compute the unfolding of the untimed support of $\mathcal{N}$ (i. e., the Petri net obtained from $\mathcal{N}$ by removing all the temporal constraints). The unfolding has finite depth when the length of the abstract processes is bounded; hence it can be computed entirely.
2. Extract the set MP of maximal processes[1]; they are the abstract processes of our PTPN $\mathcal{N}$.
3. For every $E \in MP$, construct the constraint K^p_E on the parameters of $\mathcal{N}$ under which the process is feasible.
4. Output the conjunction of the initial constraint K_0 with the negation of all constraints associated to processes which are not feasible under v:

$$K_0 \wedge \bigwedge_{E \in MP, \text{ with } v \not\models K^p_E} \neg K^p_E \,.$$

Theorem 5.3 ([ACR17]). *Let $\mathcal{N}$ be a PTPN, let v be a parameter valuation. Assume IM^{K}PO$(\mathcal{N}, v)$ terminates with result K. Then for all valuation v' of the parameters satisfying the initial constraint K_0 of the model,*

$$v' \models K \iff MaxProcesses(v'(\mathcal{N})) \subseteq MaxProcesses(v(\mathcal{N}))\,.$$

In particular $v \models K$.

Example 5.9. For the PTPN of Figure 5.3, recall that there are two maximal abstract processes $\{e_1, e_2, e_3\}$ and $\{e_2, e_3, e_0\}$. Only the first one is feasible in $v(\mathcal{N})$, i. e.,, $v \not\models K^p_{\{e_2,e_3,e_0\}}$. Then our procedure IM^{K}PO outputs the constraint

$$K_0 \wedge a_2 + a_3 + a_0 > b_1,$$

which is the negation of $K^p_{\{e_2,e_3,e_0\}}$. Remember from Definition 5.1 that K_0 is assumed to specify at least that the firing intervals are nonempty.

[1] The maximal processes can be extracted for instance by a SAT solver using an appropriate SAT encoding, or using the optimal partial-order reduction algorithm of [Rod+15].

In contrast, IM^K, would synthesize $K_0 \wedge a_2 > b_1$; therefore, the constraint synthesized by IM^{K}PO is much more permissive than the constraint output by IM^K. While IM^K requires t_1 to fire strictly before t_2, IM^{K}PO only requires that it fires before being disabled by t_0.

We now state that the output of IM^{K}PO is always equally or more permissive than the output of IM^K.

Theorem 5.4 ([ACR17]). *Let $\mathcal{N}$ be a PTPN with only finite executions, and let v_0 be a parameter valuation. Denote $K_{IM^K} = IM^K(\mathcal{N}, v)$ and $K_{IM^K PO} = IM^K PO(\mathcal{N}, v)$. Then $K_{IM^K} \subseteq K_{IM^K PO}$.*

Implementation and experiments As a proof of concept, César Rodríguez implemented both IM^K and IM^{K}PO in a prototype tool, that relies on the Cunf Petri net unfolder [RS13] to build the (untimed) unfolding of the input net, and on PolyOp (a prototype tool I implemented to support polyhedra operations, which can be seen as a wrapper around the Parma Polyhedra Library (PPL) [BHZ08]). Our implementation first enumerates all maximal (untimed) configurations, by means of an ad-hoc concretization of the Optimal Dynamic Partial-Order Reduction (ODPOR) algorithm presented in [Rod+15]. We then applied IM^K and IM^{K}PO to a few case studies modeling asynchronous circuits (see [ACR17]).

What's beyond...? Recall that IM^{K}PO can only be applied if all abstract processes are finite. This is the case of some application domains such as acyclic asynchronous (such a case study is handled in [ACR17]). In [ACR17], we also defined two variants of IM^{K}PO that (may) terminate even when abstract processes are not finite.

5.2 Non-Zeno synthesis

Model checking TAs may consist in checking whether there exists an accepting cycle (i. e., a cycle that visits infinitely often a given set of locations) in the automaton made of the product of the TA modeling the system with the TA representing a violation of the desired property. However, the existence of such an accepting cycle does not necessarily mean that the property is violated: indeed, a known problem of TAs is that they allow Zeno behaviors. An infinite run is Zeno if it takes a finite amount of time; otherwise it is non-Zeno. Zeno runs are infeasible in reality (because processors have a finite speed, and only a finite number of actions can occur in a bounded time), and thus they must be pruned during system verification. Note that the Zeno phenomenon in TAs is close to the *chattering* phenomenon in the larger class of hybrid systems (see e. g., [AC15; Alj+16]).

The problem of checking whether a timed automaton accepts at least one non-Zeno run has been tackled previously (e. g., [Tri99; TYB05; BG06; GB07; HSW13; Wan+15]). However, the synthesis of valuations satisfying some property in a PTA while guaranteeing the non-Zeno assumption had not been addressed.

We addressed in [And+17b] the synthesis of parameter valuations for which there exists at least one non-Zeno cycle in a PTA. Recall that the emptiness of the valuations set for which there exists at least one infinite run is undecidable (Theorem 3.14). The following result comes therefore without much surprise.

Lemma 5.1 ([And+17b]). *The emptiness of the valuations set for which there exists at least one non-Zeno infinite accepting run in a PTA is undecidable.*

Proof idea. Using a reasoning based on the 2-counter machine encoding proposed in [AM15]. □

5.2.1 CUB-parametric timed automata

Nevertheless, we proposed a semi-algorithm to synthesize valuations for which there exists at least one non-Zeno accepting infinite run. Just as for TAs, the parametric zone graph of PTAs (used in e. g., [Hun+02; And+09b; JLR15]) cannot be used to check whether a cycle is non-Zeno. Therefore, we introduced *clock upper bound PTAs* (CUB-PTAs), a subclass of PTAs satisfying some syntactic restrictions. First, we assume throughout this section that PTAs are enriched with an *initial parameter constraint* K_0 (similarly to our definition of PTPNs in Definition 5.1). This initial constraint can be seen as an extension of the bounded parameter domain (Definition 2.8) using linear constraints such as $p_1 \leq p_2$.

We give below an informal definition of CUB-PTAs.

Definition 5.3 (CUB-PTA [And+17b])**.** A PTA is a *CUB-PTA* if for each edge (l, g, σ, R, l'), for all clocks $x \in X$, for any parameter valuation in K_0, both the upper bound of x in g and the upper bound of x in $I(l')$ are larger or equal to the upper bound of x in $I(l)$.

Example 5.10. Consider the PTA in Figure 5.7a. This PTA is a CUB-PTA iff K_0 is such that $p_2 \geq p_1$: indeed, on the unique edge, the upper bound of x on the guard is larger or equal to the source invariant.

Whereas any TA can be transformed into a CUB-TA [Wan+15], this does not hold for PTAs.

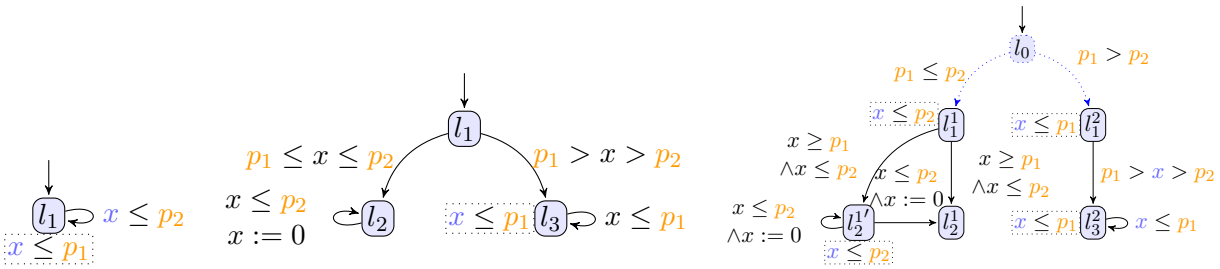
Example 5.11. No equivalent CUB-PTA exists for the PTA in Figure 5.7b, where $K_0 = \top$. Indeed, the edge from l_1 to l_2 (resp. l_3) requires $p_1 \leq p_2$ (resp. $p_1 > p_2$). It is impossible to transform this PTA into a PTA where K_0 (which is $\top$) is included in both $p_1 \leq p_2$ and $p_1 > p_2$.

However, we can use a finite union of CUB-PTAs (each of them with a different bounded parameter domain).

Proposition 5.1 ([And+17b])**.** *Any CUB-PTA can be transformed into a finite union of CUB-PTAs with the same constrained timed language.*

We can then turn a finite union of CUB-PTAs into a PTA, by adding a fresh initial location pointing to each of the initial locations of the CUB-PTAs $\mathcal{A}_i$ with an edge with guard $\mathcal{A}_i.K_0$, in 0-time with a silent transition.

Example 5.12. In Figure 5.7c (without the dotted, blue elements), two CUB-PTAs are depicted, one (say $\mathcal{A}_1$) on the left with locations superscripted by 1, and one (say $\mathcal{A}_2$) on the right superscripted with 2. Assume $\mathcal{A}_1.K_0$ is $p_1 \leq p_2$ and $\mathcal{A}_2.K_0$ is $p_1 > p_2$. Then the full Figure 5.7c (including dotted elements) is the PTA associated with the CUB-PTAs $\mathcal{A}_1$ and $\mathcal{A}_2$, and is equivalent to the PTA in Figure 5.7b.



(a) Example 1 (b) Example 2 (c) Transformed version of Figure 5.7b

Figure 5.7 – Examples: detection of and transformation into CUB-PTAs

5.2.2 Non-Zeno synthesis

Similarly to TAs, it is not possible to synthesize valuations corresponding to non-Zeno behaviors on the parametric zone graph of PTAs. However, in contrast to classical PTAs, we show that we can design a semi-algorithm synthesizing valuations for CUB-PTAs such that there exists an infinite non-Zeno cycle can be done based on (a light extension of) the parametric zone graph. The idea of the procedure is as follows [And+17b]:

1. transform the PTA into a finite union of CUB-PTAs;
2. synthesize its parametric zone graph;
3. look for strongly connected components (SCC) in the zone graph for which:
   (a) at least one location is accepting;
   (b) there is some time progress (this can be measured with an additional extra clock); and
   (c) for each clock the upper bound of which is not ∞, this clock is reset at least once in the component;
4. return the parameter valuations associated with all the aforementioned SCCs.

This procedure is a semi-algorithm as the parametric zone graph may be infinite.

Implementation The transformation of a PTA into a finite union of CUB-PTAs was implemented by Nguyễn Hoàng Gia in IMITATOR. Although the general scheme is relatively simple, the transformation is quite complex, with a lot of practical technical details. I implemented the actual synthesis algorithm, which is a simple variant of the cycle detection synthesis algorithm. We performed various experiments on a set of case studies (see [And+17b]).

What's beyond...? In [And+17b], we also provided an alternative method: instead of *transforming* a PTA into a finite union of CUB-PTAs, we can also *detect* whether a PTA already satisfies the CUB assumption for some valuations. In this latter case, one can directly apply the SCC detection, with the limitation that the result only holds for the synthesized valuations belonging to the set for which the PTA satisfies the CUB assumption.

5.3 Combining timing parameters with action parameters

In the remaining two sections, we show that the model of parametric timed automata can be extended to capture richer models. Let us first combine timing parameters with action parameters. Action parameters

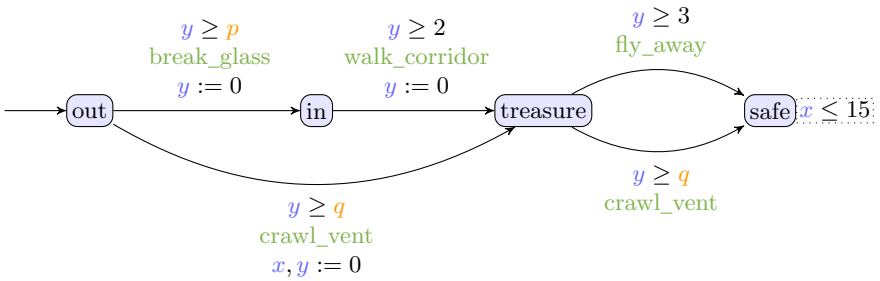


Figure 5.8 – The burglar action-controlled parametric timed automaton

can be seen as Booleans constants allowing or disallowing once for all some actions within a set of controllable actions. In [KMP15], an algorithm is proposed to compute symbolically the set of controllable actions for which a given formula (expressed in a parametric extension of CTL) is satisfied. An efficient implementation using the tool SPATULA is proposed.

In [And+16], we combined the action parameters [KMP15] with the timing parameters [AHV93] into the unified framework of action-controllable parametric timed automata (APTA). Before our work, little had been done to combine different types of parameters, typically discrete (actions) and continuous (timing) together. A notable exception is [DAr+97] where constraints are derived to ensure the correctness of the bounded retransmission protocol (BRP); one of them involves a discrete parameter (an integer-valued maximum number of retransmissions) multiplied by a continuous timing parameter. However, the procedure proposed seems to be specific to this case study.

Definition 5.4 (APTA [And+16])**.** An action-controllable PTA is a PTA enriched with a set of controllable actions.

Example 5.13. Consider the example of a burglar that wants to steal a treasure in a museum, described by the APTA in Figure 5.8. Initially, the burglar is outside (location out). First, (s)he can decide to break a window (action break_glass), which triggers an alarm at the police station. Clock x counts the time since the alarm is raised. The time necessary for the police to reach the museum is 15 minutes. Therefore, the burglar will be safe if (s)he has escaped the building before the police arrives (invariant $x \leq 15$ associated with location safe). Breaking the glass takes at least p minutes. Once the window is broken, the burglar is in and takes at least 2 minutes to walk to the treasure room (action walk_corridor). The other possibility to reach the treasure is to crawl in a ventilation (action crawl_vent) which has the advantage of not triggering the alarm until in the treasure room. This takes at least q minutes. To escape the treasure room, the burglar can be picked up by a friend flying an helicopter (action fly_away), after running to the roof of the building in at least 3 minutes. Or else, (s)he can decide to crawl back through the ventilation system.

We considered only the reachability problem in [And+16], i. e., the synthesis of action and timing parameters for which a discrete location is reachable. The following result trivially comes from the fact that the class of APTAs can be seen as an extension of PTAs.

Lemma 5.2 ([And+16])**.** *The EF-emptiness problem is undecidable for APTAs.*

Nevertheless, we proposed a semi-algorithm for synthesizing the set of actions and timing parameters (in the form of a set of linear constraints on controllable actions seen as Booleans, and timing parameters) for which a given location is reachable. Our procedure can be summarized as follows [And+16]:

1. given an APTA, we enable all controllable actions, which gives a PTA $\mathcal{A}$;
2. we compute a mixed transition system, which is essentially the parametric zone graph $\mathcal{PZG}$ of $\mathcal{A}$;
3. for each target state in $\mathcal{PZG}$, we synthesize using techniques from [KMP15] the constraint on the controllable actions such that this target state is reachable in $\mathcal{PZG}$, and we return the conjunction of the timing parameter constraint and the action parameter constraint;
4. we return the union of all the aforementioned computed constraints.

Example 5.14. Consider again the burglar example in Figure 5.8. The constraint for which the safe location is reachable (which is the unsafe location from the point of view of the museum!) is

$$\begin{array}{ll} & \text{break\_glass} \wedge \text{walk\_corridor} \wedge \text{fly\_away} \wedge p \leq 10 \\ \vee & \text{crawl\_vent} \wedge q \leq 15 \end{array}$$

The system designers can therefore disable some actions and/or tuning some timing parameters so as to make the system safe. Examples of possibilities to make the treasure safe can be:

- disabling crawl_vent (e. g., by installing some proper bars) and fly_away (e. g., by installing some net on the rooftop); or
- disabling break_glass (e. g., by installing an armored window) and tuning $q > 15$ (e. g., by increasing the length of the ventilation pipes); or
- tuning $p > 10$ (e. g., by installing a solid enough window) and $q > 15$.

Implementation Our implementation is made of IMITATOR linked to SPATULA with a Python script. In [And+16], we applied our toolkit to a sample example of a railway gate controller. Experiments show that our method outperforms exhaustive enumeration, even when the timing parameter domain is finite (e. g., bounded integers).

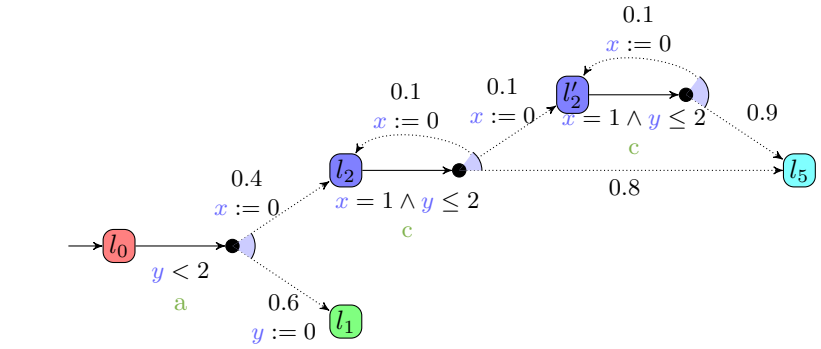
Future works include the extension to a larger subset of TCTL, at least AF—which is not trivial though.

5.4 Parameter synthesis in probabilistic models

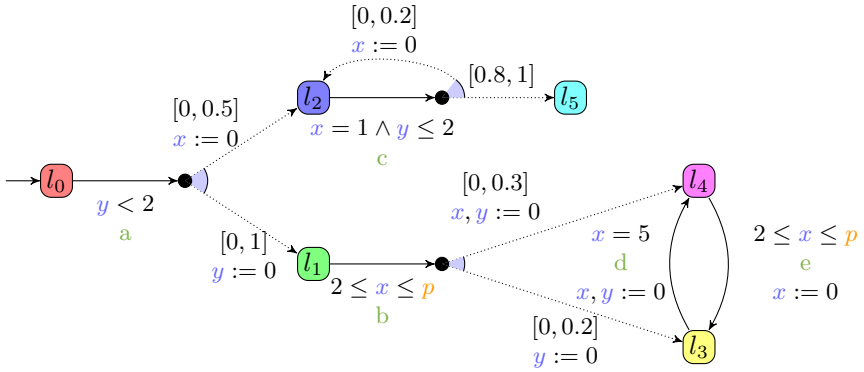
Another natural way to extend parametric timed automata is to add probabilities. The setting of timing parameters with (non-parametric) probabilities was considered in some earlier works, e. g., [Cha+08; AFS13b; JK14].

Here, we consider a slightly different setting, inspired by interval Markov chains [JL91], a formalism that takes into account imprecision in the transition probabilities. This formalism extends Markov chains by allowing to specify *intervals* of possible probabilities on transitions instead of exact values. Methods have then been developed to decide whether there exists Markov Chains with concrete probability values that match the specified intervals [Del+12].

In [AD16], we combined both abstraction approaches into a single specification theory: Parametric Interval Probabilistic Timed Automata (PI$\mathbb{P}$TAs for short). In this setting, parameters can be used in order to abstract timed constants on transition guards while intervals can be used to abstract imprecise transition probabilities. It is important to be able to decide whether the probability intervals that are specified in a model allow defining consistent probability distributions (i. e., can be matched in a real-life implementation). This is called the *consistency* problem.



(a) A $\mathbb{P}$TA



(b) An example of PI$\mathbb{P}$TA

Figure 5.9 – Examples of I$\mathbb{P}$TA and PI$\mathbb{P}$TA

Given an arbitrary set S, we call an *interval distribution* over S a function Υ that assigns to each element of S an interval of probabilities $[a, b] \subseteq [0, 1]$. Let $\mathsf{Int}_{[0,1]}(S)$ denote the set of all interval distributions over S.

Definition 5.5 (PI$\mathbb{P}$TA [AD16])**.** A Parametric Interval Probabilistic Timed Automaton (PI$\mathbb{P}$TA) is a parametric timed automaton without invariant, and with edges of the form (l, g, σ, Υ), where $l \in L$, g is a guard, $\sigma \in \Sigma$, and $\Upsilon \in \mathsf{Int}_{[0,1]}(2^X \times L)$ is an interval distribution.

An interval probabilistic timed automaton (I$\mathbb{P}$TA) is a PI$\mathbb{P}$TA without timing parameters. A probabilistic timed automaton ($\mathbb{P}$TA) is an I$\mathbb{P}$TA with punctual distributions (i. e., intervals reduced to a point).

Example 5.15. Figure 5.9a presents an example of a $\mathbb{P}$TA with two clocks x and y. For example, l_0 can be exited whenever $y < 2$; then, with probability 0.4 the target location becomes l_2, resetting x; or with probability 0.6 the target location is l_1, resetting y. The transition from l_2 can be explained similarly.

Once a parameter valuation is fixed, the resulting I$\mathbb{P}$TA represents a potentially infinite set of $\mathbb{P}$TAs. In order to relate a given I$\mathbb{P}$TA with the $\mathbb{P}$TAs it represents, we use the notion of *implementation* defined hereafter. This notion is similar to the one defined in the context of (parametric) Interval Markov Chains [DLP16]. Remark that a $\mathbb{P}$TA implementing an I$\mathbb{P}$TA needs to conserve the exact same clocks, guards and resets. This definition is rather technical, and is central is the correctness of our framework.

Definition 5.6 (Implementation of an IℙTA)**.** Let $\mathcal{P} = (\Sigma, L, l_0, X, prob)$ be a ℙTA and $\mathcal{IP} = (\Sigma, L', l'_0, X, \mathbb{I})$ be an IℙTA.

We say that $\mathcal{P}$ is an implementation of $\mathcal{IP}$, written $\mathcal{P} \models \mathcal{IP}$, iff there exists a relation $\mathcal{R}_P \subseteq L' \times L$, called an *implementation relation* $(l'_0, l_0) \in \mathcal{R}_P$ and, whenever $(l', l) \in \mathcal{R}_P$, we have

- $\forall (l', g', \sigma, \upsilon) \in prob, \exists (l, g', \sigma, \Upsilon) \in \mathbb{I} \ \ \upsilon \preceq_{\mathcal{R}_P} \Upsilon$, and
- $\forall (l, g, \sigma, \Upsilon) \in \mathbb{I}, \exists (l', g, \sigma, \upsilon) \in prob \ \ \upsilon \preceq_{\mathcal{R}_P} \Upsilon$,

where $\upsilon \preceq_{\mathcal{R}_P} \Upsilon$ iff $\exists \delta \in \mathsf{Dist}(L' \times L)$

- $\forall (R', l') \in 2^X \times L', \upsilon(R', l') > 0 \Rightarrow \sum_{l \in L} (\delta(l', l)) = 1$,
- $\forall (R, l) \in 2^X \times L, \sum_{l' \in L'} (\upsilon(R, l') \cdot \delta(l', l)) \in \Upsilon(R, l)$, and
- $\delta(l', l) > 0 \Rightarrow (l', l) \in \mathcal{R}_P$.

Example 5.16. Consider the PIℙTA $\mathcal{PIP}$ given in Figure 5.9b. When a distribution is made of a single target location with probability 1, we simply omit the distribution (e. g., between l_3 to l_4).

First, observe that $\mathcal{PIP}$ cannot be consistent if the edge labeled with b can be taken: indeed, no implementation of the intervals $[0, 0.3]$ and $[0, 0.2]$ is such their sum is equal to 1.

Now, let v_1 be the parameter valuation such that $v_1(p) = 1$. In the IℙTA $v_1(\mathcal{PIP})$, the transition outgoing from l_1 can never be taken, as its guard becomes $2 \leq x \leq 1$, which is unsatisfiable. Then, it is clear that the ℙTA $\mathcal{P}$ given in Figure 5.9a is an implementation of $v_1(\mathcal{PIP})$. As a consequence, $v_1(\mathcal{PIP})$ is a consistent IℙTA.

5.4.1 Consistency in interval probabilistic timed automata

First, in the context of interval probabilistic timed automata with no timing parameters (IℙTAs), we propose an algorithm that solves the consistency problem, i. e., the existence of at least an implementation.

Theorem 5.5 ([AD16])**.** *The consistency problem is decidable for IℙTAs.*

Proof idea. The proof requires several results. First, we define the symbolic semantics of an IℙTA as an interval Markov chain. Then, we define a consistency definition for interval Markov chains that matches our consistency definition for IℙTAs. We then show that there exists an implementation of an interval Markov chain iff there exists one with the same structure. We then prove that an IℙTA is consistent iff its probabilistic zone graph is consistent. The proof of this latter result requires to reconstruct an IℙTA from a zone graph. Finally, we introduce an algorithm for checking the consistency of interval Markov chains, in the lines of [Del15]. □

5.4.2 Consistency synthesis in parametric interval probabilistic timed automata

In the parametric setting, the following problem asking for the existence of a parameter valuation for which the resulting IℙTA is consistent is undecidable.

Theorem 5.6 ([AD16]). *The consistency-emptiness problem is undecidable for PIPTAs.*

Proof idea. Almost immediate from the undecidability of the EF-problem for PTAs (we reused the 2-counter machine encoding of [Ben+15]). □

Now, we retrieve the decidability if we add the restriction that the parameters set must be partitioned into lower-bound parameters and upper-bound parameters.

Theorem 5.7 ([AD16]). *The consistency-emptiness problem is decidable for L/U-PIPTAs.*

Proof idea. The proof relies on the monotonicity property of L/U-PIPTAs, requires to rebuild an L/U-PTA from the zone graph of the PIPTA, and finally reuses the decidability of the EF-universality problem for L/U-PTAs [AL17a]. □

Finally, we proposed a semi-algorithm that synthesizes parameters for which a PIPTA becomes consistent (see [AD16] for details).

Example 5.17. Consider again the PIPTA in Figure 5.9b. The set of valuations for which the resulting IPTA is consistent is $p < 2$.

5.5 Perspectives

Robustness Let us put our definition of precise robustness into perspective with other works on robustness, where decidability is obtained, such as [San11; Bri+13; SBM14] for timed automata or [Aks+16] for time Petri nets.

On the one hand, our definition is more precise, as it allows to measure the variability in several dimensions w.r.t. the untimed language, or the untimed language up to partial orders. In contrast, in these previous works, the timing constants can all vary in only one direction (enlarging or shrinking) and with the same variability (or a "vector of variations"), while our definition that replaces timing constants with parameters allows to consider both shrinking and enlarging, and allows to have some parameters not varying, which is usually not allowed in the literature (in this case, the system is simply considered non-robust).

On the other hand, our algorithms are not guaranteed to terminate in contrast to [San11; Bri+13; SBM14] (the case of [Aks+16] is different as the restriction to ensure decidability seems very strong).

Future works should take advantage of the decidability results for the classes of L/U-PTAs and U-PTAs: these classes can be seen as formalisms modeling multidimensional guard enlargement and shrinking. While some problems are undecidable for L/U-PTAs (e. g., robust preservation of the untimed language [AM15]), some syntactic restrictions (e. g., U-PTAs) could be added to ensure decidability of problems related to a more precise definition of robustness than the usual unidimensional one.

Decision problem Concerning parametric timed Petri nets, the language-preservation-emptiness was not proved undecidable (contrarily to what we achieved for PTAs). While we suspect this would be the case too, the demonstration does not seem to be easy, and the proof remains to be done. The same problem up to partial orders would be an interesting theoretical challenge.

Multiview parameter synthesis Formalisms mixing timing parameters and probabilities are very expressive, and problems are highly challenging. Adding probabilistic parameters (with or without timing parameters) to our parametric interval probabilistic timed automata would open interesting perspectives.

Chapter 6

Application to parametric schedulability

Beyond the theoretical aspects of Chapter 3 and the algorithmic aspects of Chapters 4 and 5, parametric timed model checking can be used for solving practical problems. In this chapter, we focus notably on schedulability analysis for real-time systems under uncertainty, i. e., where some timing constants may be known with a limited precision or completely unknown.

We propose techniques to perform parametric schedulability analysis with parametric timed automata and compare them to analytical methods (Section 6.1). We then introduce the formalism of parametric task automata (Section 6.2). Finally, we describe our solution to the industrial FMTV challenge by Thales (Section 6.3).

6.1 Parametric schedulability analysis

The schedulability problem Real-time systems are systems for which computing a correct result is important, but also for which computing this result in a *timely* manner is just as important as the correctness. Critical real-time systems are real-time systems for which failing to compute a result in a timely manner may result in dramatic consequences, such as high financial costs, or loss of human lives.

A real-time system comprises of a set of tasks, to be executed on one processor ("uniprocessor") or several processors ("multiprocessor"). Tasks generally feature a period (i. e., the time every which occurrence an instance of the task is activated), a relative deadline (i. e., the time after the activation by which the instance must complete) and an execution time (possibly in the form of a best-case and a worst-case execution time).

A *scheduler* decides which task to execute at a given time. Common schedulers include fixed-priority scheduling (each task is statically assigned a priority, and whenever a task is terminated, the highest priority task in the queue will be selected) or earliest-deadline first (the task with the most urgent deadline is selected first). Schedulers can possibly be preemptive, i. e., temporarily interrupt a lower-priority task in order to execute a higher-priority task (and then resume the execution of the lower-priority task later).

In order to guarantee the timely answer in a critical real-time system, a *schedulability analysis* is absolutely necessary. The schedulability analysis consists in verifying statically (before the execution of the system) whether the system is schedulable; or, in other words, for any execution, all task instances will always be completed before their relative deadline, according to the scheduler.

Since the seminal work by Liu and Layland [LL73] on scheduling real-time systems, many works addressed the schedulability in various settings, notably on uniprocessor environments: for deadline-monotonic scheduling policies without [LW82] or with [Aud91] offsets, for earliest-deadline first with various assumptions on the deadlines, periods and offsets [LL73; BRH90; Spu96] and with shared re-

sources [CL90; Bak91], or in the presence of precedence constraints (e. g., some task instance must be completed before an instance of another task is executed) [CSB90; For+10; For+11].

6.1.1 Schedulability analysis using parametric stopwatch automata

In [AM01; AM02], timed automata and stopwatch automata are used to perform schedulability analysis. Stopwatch automata extend timed automata with the power of stopping the elapsing of some clocks in some locations. This feature makes the model very expressive, and the mere reachability problem for TAs becomes undecidable [CL00].

In [CPR08], PTAs are used to perform parametric schedulability analysis: whereas the general case is unsurprisingly undecidable, the authors exhibit a subclass for which the schedulability-synthesis (i. e., synthesizing all valuations for which the system is schedulable) can be performed exactly.

In [Fri+12], the goal was to perform parametric schedulability analysis, i. e., synthesizing the timing constants seen as parameters (that can be deadlines, periods…) so that the system becomes schedulable for a given scheduling policy. They proposed an *ad-hoc* method consisting in running the behavioral cartography BC [AF10] on parametric stopwatch automata. Their approach was then applied to a prospective architecture for the flight control system of the next generation of spacecrafts designed at ASTRIUM Space Transportation.

While Giuseppe Lipari was an MSC fellow in LSV, ENS Cachan, we started to work together on schedulability analysis, together with Laurent Fribourg, Romain Soulat and Sun Youcheng. In [Sun+13b], we proposed a more systematic method for analyzing the schedulability of preemptive fixed priority real-time distributed systems. More precisely, we considered applications modeled by a set of *pipelines* (also called *transactions* in [PG98]), where each pipeline is a sequence of periodic tasks to be executed in order, and all tasks in a pipeline must complete before an end-to-end deadline. All processors in the distributed system are connected by one or more CAN bus [Dav+07], a network standard used in automotive applications. A pipeline is assigned two fixed elements: T^j is the pipeline period and D^j_{e2e} is the end-to-end deadline. This means that all tasks of the pipeline are activated together every T^j units of time; and all tasks should be completed within a time interval of D^j_{e2e}. Each task t_i is characterized by the following elements:

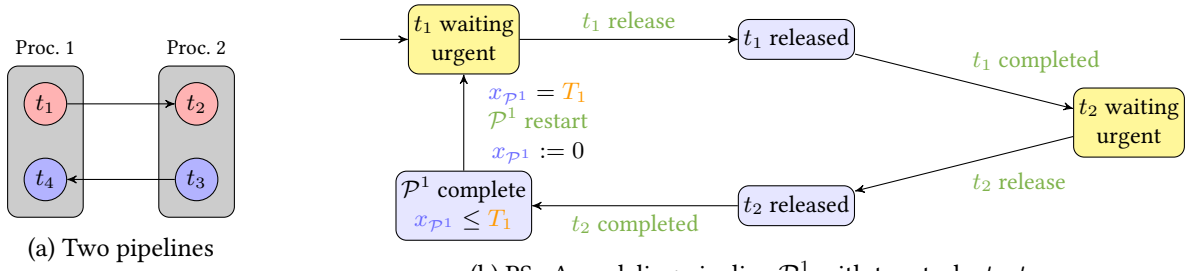
- a period T_i;
- a deadline D_i;
- a worst-case computation time W_i;
- a processor on which to execute; and
- a priority.

The deadlines, periods and worst-case computation times are all considered to be (potentially) unknown, i. e., *parameters*.

Example 6.1. Consider the example in Figure 6.1a of two pipelines $\mathcal{P}^1, \mathcal{P}^2$ with $\mathcal{P}^1 = \{t_1, t_2\}, \mathcal{P}^2 = \{t_3, t_4\}$, with t_1 and t_4 to be executed on processor 1, while t_2 and t_3 are to be executed on processor 2. We assume that t_1 has higher priority than t_4, and that t_3 has higher priority than t_2.

We are concerned with the following goal of schedulability synthesis: *for what values of the timing constants (periods, deadlines…) seen as parameters is the system schedulable?*

To achieve this goal, we define a natural extension of PTAs:



(a) Two pipelines

(b) PSwA modeling pipeline $\mathcal{P}^1$ with two tasks t_1, t_2

Figure 6.1 – Two pipelines, and the modeling of one of them as a parametric stopwatch automaton

Definition 6.1 (Parametric stopwatch automata [Fri+12; Sun+13b])**.** A parametric stopwatch automaton (PSwA) is a PTA where the elapsing of some clocks can be stopped in some locations.

Our approach for schedulability synthesis can be summarized as follows [Sun+13b]:

1. we transform each pipeline into a PSwA;
2. we transform the scheduler of each processor into a PSwA;
3. we build a PSwA resulting from the parallel composition of all aforementioned PSwAs, synchronizing on actions corresponding to tasks releases and completions;
4. we reduce schedulability synthesis to (the negation of the) EF-synthesis of a failure location (corresponding to a deadline miss) of that PSwA.

(In fact, in [Sun+13b], for mainly practical reasons—EFsynth was not implemented in IMITATOR at the time of that work—, we used instead repeated calls to the inverse method IM.)

Example 6.2. Consider again the pipelines example in Example 6.1. Figure 6.1b shows the PSwA model of pipeline $\mathcal{P}^1$. Urgent locations (labeled with the keyword "urgent") are locations in which time cannot elapse: they can be encoded using an additional clock set to be 0 when entering the location, and constrained to be equal to 0 in the location invariant (some tools encode urgent locations natively—this is the case of IMITATOR).

Figure 6.2 shows the PSwA model of the preemptive processor 2 (on which tasks t_1 and t_4 shall execute). Recall that task t_1 has higher priority over task t_4. The keyword "stopped" in a location denotes the set of clocks to be stopped in that location. The processor starts by being *idle*, waiting for a task release. As soon as a request has been received (e. g., action "t_4 release"), it moves to one of the states where the corresponding task is running ("t_4 running"). If it receives another release request ("t_1 release"), it moves to the location corresponding to the higher priority task running ("t_1 release, t_4 released"). The fact that t_1 does not execute anymore is modeled by the blocking of the clock x_{t_4} corresponding to task t_4. This is where preemption is needed. Moreover, while a task executes, the scheduler automaton checks if the corresponding pipeline misses its deadline (e. g., guard $x_{\mathcal{P}^1} > D^1_{e2e}$, where D^1_{e2e} is the pipeline deadline). In the case of a deadline miss, the processor moves to a special failure location ("deadline missed").

6.1.2 Experiments and comparison

We implemented our approach in IMITATOR and compared it with two other tools:

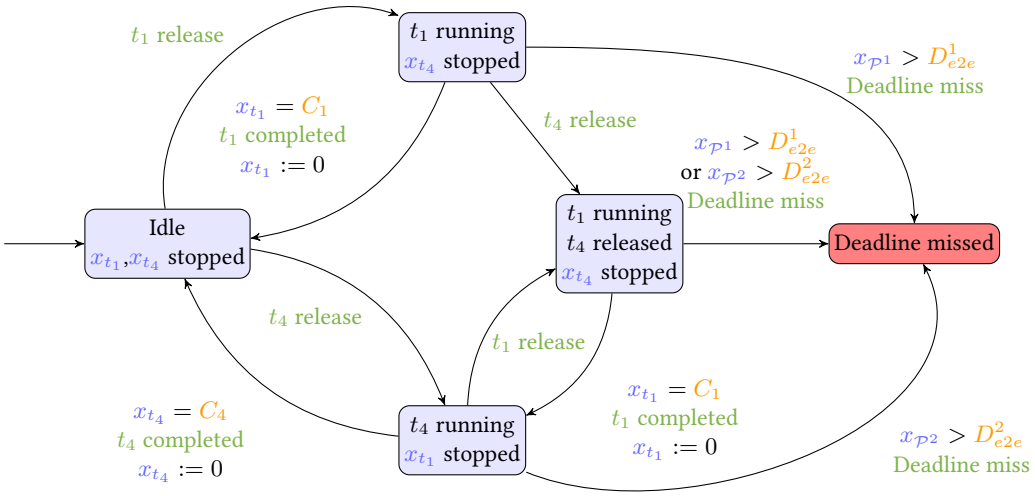


Figure 6.2 – PSwA modeling the preemptive processor 2 with two tasks t_1, t_4

- MAST [Gon+01] is implemented and maintained by the CTR group at the *Universidad de Cantabria* and performs schedulability analysis for distributed real-time systems; and
- RTSCAN is a prototype tool implementing an analytical method proposed in [Sun+13b] by Sun Youcheng and Giuseppe Lipari that extends the test proposed in [SLS98] .

Note that the results output by MAST and RTSCAN are integer-valued, whereas IMITATOR synthesizes dense constraints over rational-valued parameters; therefore only IMITATOR can give a measure of the system robustness.

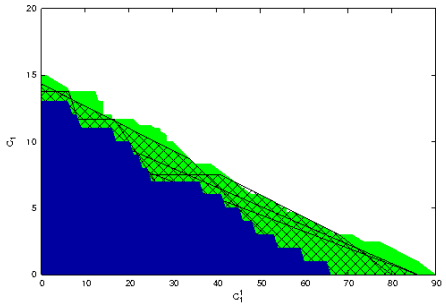
We ran two test cases:

1. The first test case [PG98] consists of three simple periodic tasks and one pipeline, running on two processors, connected by a CAN bus. The pipeline models a remote procedure call from processor 1 to processor 3. All tasks have deadlines equal to periods, and also the pipeline has end-to-end deadline equal to its period.
2. The second test case [Wan+06] consists of two pipelines on three processors and one network, where pipeline P^1 is periodic with period 200 ms and end-to-end deadline equal to the period.

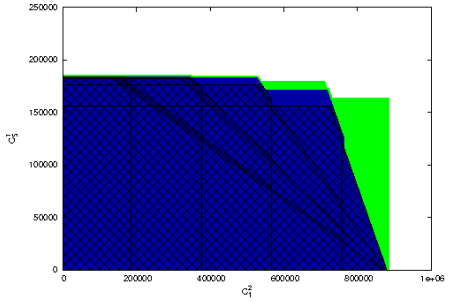
Full details about the test cases can be found in [Sun+13b].

In order to simplify the visualization of the results, for each test case we present the schedulability region generated for two parameters only. However, all three methods are general and can be applied to any number of parameters.

Results We give the results of our experiments in Figure 6.3. Without surprise, as it synthesizes an exact result when it terminates (which is the case here), the schedulability region computed by IMITATOR dominates the other two tools.



(a) Test case 1



(b) Test case 2

Figure 6.3 – Schedulability regions produced by RTSCAN (hatched), MAST (dark blue, below), and IMITATOR (light green, above)

6.2 Parametric task automata: A unified formalism for uniprocessor schedulability

In [And17], I introduced parametric task automata (PTaskA), as a parametric extension of task automata introduced in [NWY99; Fer+07]. Parametric task automata allow a compact representation of a real-time system, and can be seen as a unified formalism to model uniprocessor schedulability problems with several types of tasks (periodic, sporadic, or more complex). Most types of schedulers, including EDF (earliest-deadline first), FPS (fixed-priority) and SJF (shortest job first), with or without preemption, can be used. Most importantly, uncertain or unknown timing constants can be used thanks to timing parameters.

A PTaskA is essentially a PTA enriched with tasks, an instance of which is activated every time the PTA enters a location. Let $\mathcal{T} = \{t_1, t_2, \cdots\}$ be a set of tasks. Each task is characterized by three *timings*, i. e., constants in $P \cup \mathbb{Q}_+$:

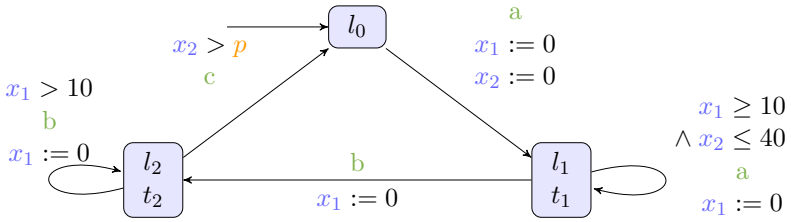
1. B: its best-case execution time,
2. W: its worst-case execution time, and
3. D: its relative deadline (i. e., the latest time after the release of the task by which it must be completed).

Given a task t and a parameter valuation v, we denote by $v(t)$ the task where the parameters in the timings (i. e., B, W and D) are replaced with their value in v.

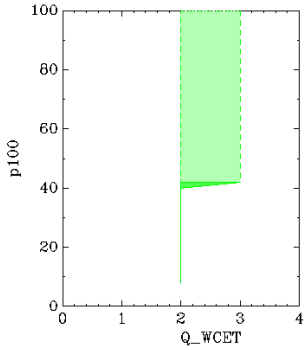
Each task can have several *instances*, i. e., copies of the same task. An instance of task t is written (t, b, w, d) where $b \in \mathbb{R}_+$ (resp. $w \in \mathbb{R}_+$) is the best-case (resp. worst-case) remaining computation time, and $d \in \mathbb{R}_+$ the remaining time before the deadline.

Definition 6.2 (PTaskA [And17])**.** A *parametric task automaton* is a PTA extended with

1. $\mathcal{T}$: a set of tasks, and
2. $T : L \rightharpoonup \mathcal{T}$: a partial task function, assigning to some locations a task.



(a) An example of a PTaskA



(b) Schedulability region

Figure 6.4 – A parametric task automaton and its schedulability region

Example 6.3. Figure 6.4a (coming from [And17] and inspired by [Fer+07, Fig.1]) describes a PTaskA with 2 clocks, and 2 tasks: t_1, an instance of which is activated every time the PTaskA enters l_1, and t_2 (in l_2). For t_1, we have $B = 1$, $W = 2$ and $D = 10$; for t_2, $B = 2$, $W = p'$ and $D = 8$. Note that our formalism allows one to define parameters both in the automaton (p) and the task timings (p').

Basically, this PTaskA can create in l_1 between 1 and 5 instances of t_1 (but no more frequently than every 10 time units); then, it moves to l_2 where it can remain as long as wished, creating instances of t_2 (again no more frequently than every 10 time units). Eventually, the PTaskA can move back to the initial location no sooner than p time units since the entering of l_1.

Intuitively, this PTaskA will be schedulable only if p' (W of t_2) is not too large, and only when p is not too small (otherwise one may loop too fast through the automaton for all tasks to terminate before their deadline).

Thanks to the expressive power of PTAs, this formalism is richer than the traditional periodic tasks (characterized by their period) or sporadic tasks (characterized only by their minimal inter-arrival time).

We address here the following problem.

Schedulability-emptiness problem:
INPUT: A PTaskA $\mathcal{A}$ and a scheduling strategy Sch
PROBLEM: is the set of valuations v for which $v(\mathcal{A})$ is schedulable for strategy Sch empty?

6.2.1 Decidability and undecidability

The following result comes immediately on the one hand from the undecidability of the EF-emptiness problem for PTAs [AHV93] and on the other hand from the undecidability of the schedulability problem for task automata [Fer+07].

Theorem 6.1 ([And17]). *The schedulability-emptiness problem is undecidable for PTaskA with at least three parametric clocks and a single timing parameter, whatever the scheduling strategy.*

The schedulability-emptiness problem is undecidable for general PTaskA.

In the non-parametric setting, the number of instances of a task t (with timings B, W, D) is intuitively bounded by $\lceil D/W \rceil$; indeed, when the number of instances exceeds this bound, the queue will be overflown in the sense that it will be impossible to finish that many instances before the deadline D. Therefore, as

soon as the queue exceeds this value, the system is non-schedulable and therefore, it is sufficient to consider a bounded queue for schedulability analysis. However, this reasoning does not hold for general PTaskA, as W can be arbitrarily small, and D arbitrarily large. This motivates the following definition.

Definition 6.3. A PTaskA has *schedulable-bounded parameters* if, for each task t, its worst-case execution time W is bounded in $[a, \infty)$ or $[a, b]$ with $a > 0$, and its deadline D is bounded in $[a, b]$, with $a, b \geq 0$.

That is, the W cannot be 0, and the deadline cannot be infinite. Therefore, the maximum number of instances to be considered for a task is bounded by $\lceil \max(D) / \min(W) \rceil$, where $\max$ (resp. $\min$) denotes the upper (resp. lower) bound of a parameter.

Example 6.4. The PTaskA in Figure 6.4a trivially meets the schedulable-boundedness assumption, as necessarily $p' \geq B = 2 > 0$. In addition, the maximum number of instances necessary to check schedulability is $10/2 = 5$ for t_1 and $8/2 = 4$ for t_2.

We then slightly restrain the use of parameters in PTaskA in the following definition, following the similar restriction in L/U-PTAs.

Definition 6.4. A PTaskA is an *L/U-PTaskA* if its parameters set is partitioned into lower-bound parameters and upper-bound parameters.

We then obtain the following decidability result.

Theorem 6.2 ([And17])**.** *The schedulability-emptiness problem is decidable for L/U-PTaskAs with schedulable-bounded parameters, for non-preemptive* FPS *and* SJF*, and non-preemptive* EDF *without parametric deadlines.*

Proof idea. The proof works in two steps. First, we prove that the non-preemptive scheduler can be encoded into an L/U-PTA. Second, following an encoding in [Fer+07], we transform the actual task automaton into another L/U-PTA—which is possible from the assumptions of the schedulable-bounded parameters and the non-preemptive scheduling. Then we reduce the schedulability-emptiness to the EF-emptiness problem, which is decidable for L/U-PTAs. □

6.2.2 Synthesis

We can adopt a more pragmatical view. Since we only constrain a scheduler to be encoded using a stopwatch automaton, we therefore directly translate any scheduler (preemptive or not) into a (parametric) stopwatch automaton. Even in the decidable cases (where we showed that stopwatches are not needed), we potentially use stopwatches.

General idea. We will consider the synchronous product of two PSwAs in parallel: the actual PTaskA $\mathcal{A}$, and the translation of the scheduler Sch into a second PSwA $\mathcal{A}_{\mathsf{enc}}(\mathsf{Sch})$. As noted earlier, a PTaskA is just a PTA, where some locations activate task instances. Therefore, the PTaskA can be transformed into an almost-identical PSwA (without stopwatches), by labeling each edge going into a location where task t is

activated by a fresh action activating t. Then, the scheduler will synchronize on these activation actions, and manage the tasks queue according to its strategy.

The locations of $\mathcal{A}_{\mathsf{enc}}(\mathsf{Sch})$ are all possible configurations of the discrete part of the tasks queue, of which there is a finite number thanks to the schedulable-boundedness assumption. At any time, if the size of the queue overflows the maximal queue size implied by the schedulable-boundedness assumption, $\mathcal{A}_{\mathsf{enc}}(\mathsf{Sch})$ will go to a special error location, which denotes that the system is non-schedulable.

Using the above construction $\mathcal{A}_{\mathsf{enc}}$, we have:

Proposition 6.1 ([And17]). *Given a PTaskA $\mathcal{A}$ and a scheduling strategy* Sch*, the system is schedulable exactly for the parameter valuations for which the error location is unreachable in $\mathcal{A} \parallel \mathcal{A}_{\mathsf{enc}}(\mathsf{Sch})$.*

Implementation

As writing such a scheduler quickly becomes tedious and error-prone, we implemented an external program (650 lines of Python) that takes as input on the one hand a scheduling strategy Sch and on the other hand the list of tasks of the PTaskA $\mathcal{A}$ (with their timings, their priority (for FPS), their maximum number of instances…), and automatically generates the corresponding PSwA $\mathcal{A}_{\mathsf{enc}}(\mathsf{Sch})$ in the IMITATOR input format. Then, it suffices to pass to IMITATOR the model made of $\mathcal{A}$ and $\mathcal{A}_{\mathsf{enc}}(\mathsf{Sch})$.

We demonstrated the expressive power of our formalism on several examples, allowing also for robust schedulability (see [And17]).

Example 6.5. Let us go back to Figure 6.4a, and assume a preemptive FPS scheduler. First, we set $p = 100$, and we obtain that the system is schedulable for $p' \in [2, 3]$. Second, we set $p' = 3$, and we obtain that the system is schedulable for $p \geq 42$. This confirms both intuitions that p' should be not too large, and p large enough for the system to be schedulable. Finally, we run an analysis with both parameter dimensions, which gives:

$$p' \in [2, 3] \wedge p \geq 42 \quad \vee \quad p' = 2 \wedge p \in [8, 42) \quad \vee \quad p' > 2 \wedge p < 42 \wedge p \geq 36 + 2 \times p'$$

A graphical representation output by IMITATOR is given in Figure 6.4b (where `p100` stands for p and `Q_WCET` for p').

6.3 The Thales FMTV challenge

In 2014, Thales R&D published an open industrial challenge in the framework of the Formal Methods for Timing Verification workshop (FMTV 2015).[1]

6.3.1 Challenge description

The description of the challenge was done in the form of a picture (given in Figure 6.5), together with some additional explanations in English (ommited here). There are four tasks T1, T2, T3 and T4, distributed in different processing units and performing respective functionalities. The task T1 periodically receives frames from the camera and pre-processes them. Task T2 embeds further tracking information into the video frame pre-processes by Task T1. Task T2 then inserts the video frame into a register, denoted as Register23. Then Task T3 reads the frame from the register, removes the noise and tries to put the resulting

[1] The full (informal!) specification can be found at `http://waters2015.inria.fr/challenge`.

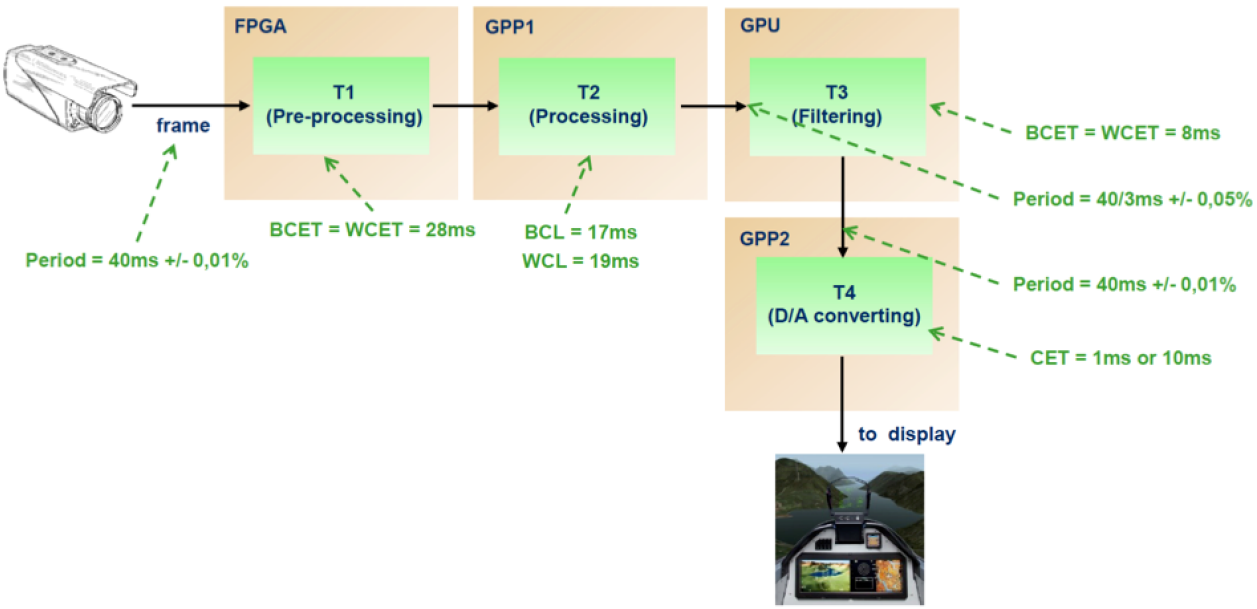


Figure 6.5 – The FMTV challenge as described by Thales

video frame into a buffer, denoted as Buffer34. In the end, Task T4 reads frames from the buffer, converts them from digital to analogue and sends the final frame to the display.

Tasks T1, T3 and T4 are periodic, but their triggering clocks are subject to drift. That is, their periods P_1, P_3 and P_4 are unknown constants. More specifically, $P_1 \in [40 - 40 \times 0.01\%, 40 + 40 \times 0.01\%]$ ms, $P_3 \in [\frac{40}{3} - \frac{40}{3} \times 0.05\%, \frac{40}{3} + \frac{40}{3} \times 0.05\%]$, and $P_4 \in [40 - 40 \times 0.01\%, 40 + 40 \times 0.01\%]$ ms. Task T2 is triggered by the completion of T1.

Each task has its Best-Case and Worst-Case Execution Time (BCET and WCET) or Latency (BCL and WCL): $BCET_1 = WCET_1 = 28$ ms, $BCL_2 = 17$ ms, $WCL_2 = 19$ ms, $BCET_3 = WCET_3 = 8$ ms. As for task T4, when it reads Buffer34 and there is no frame within the buffer, it performs an empty cycle with execution 1ms; otherwise, it executes 10 ms and sends the result to display.

Challenge:
Compute the minimum and maximum latencies for a given frame from the camera output to the display input, for a buffer size $n = 1$ and $n = 3$.

6.3.2 Solution using IMITATOR

Although the challenge is not explicitly a parametric model checking problem, it does feature parameters: the uncertainty in the periods are not jitters, but *constant uncertain periods*. The informal specification is clear about that point: the period is not fully known, but may not vary during one execution. This is exactly the definition of a timing parameter—that is, an *unknown constant*.

We proposed a solution in [ALS15] using parametric timed automata and IMITATOR.

Our key element in the solution is to consider a single *arbitrary* frame processing. Thanks to the symbolic representation offered by IMITATOR, we can start from an arbitrary state, and perform a finite number of actions simulating this arbitrary frame. Measuring the time from its input to the output, we will therefore obtain a (parametric) best and worst case time. We use parameters (i. e., unknown constants) to model the uncertain periods; we also use an additional parameter $E2E \geq 0$ which represents the end-to-end latency of the target frame.

In our modeling, the period of Task T3 is a *parameter* $P3\_uncertain$, initialized as follows:

$$P3\_uncertain \in [40 - \text{P3\_delta}, 40 + \text{P3\_delta}]$$

where $\text{P3\_delta} = 0.05\,\% \times 40 = \frac{1}{150}$. Recall that parameters in PTA are unknown *constants*, i. e., the value of which cannot evolve during the execution; this is exactly what we need to model $P3\_uncertain$.

Similarly, the period of Task T4 is a *parameter* $P4\_uncertain$, initialized as follows:

$$P4\_uncertain \in [40 - \text{P4\_delta}, 40 + \text{P4\_delta}]$$

where $\text{P4\_delta} = 0.01\,\% \times \text{P4} = 0.004$.

At first, we solve the case with $n = 1$ for Buffer34. We extensively used the features offered by IMITATOR, and notably the discrete variables, which are rational-valued global variables that can be read in guards and invariants, and modified along transitions (they are mostly syntactic sugar for extra locations, but greatly ease the modeling).

Modeling Camera, Task T1, Task T2 First, let us explain the camera, and tasks T1 and T2. In order to reduce the state space, we model the camera, Task T1 and Task T2 into a single PTA. We also use this PTA to non-deterministically initialize the buffer and the frame currently processed by Task T4.

We choose an arbitrary frame with index target for end-to-end latency estimation and *we start from the exact point such that the target frame is handled from Task T1 to task T2.* A clock $ckT1T2$ is initialized to be WCET_1 and measures the end-to-end latency of target frame.

We do not model the period of the camera (or task 1), since we are only interested in a single frame. The buffer is modeled using discrete variables.

Task T3 Task T3 is modeled by a periodic PTA. At the initial point, the PTA T3 is non-deterministically waiting for a new activation or executing. When T3 finishes execution, it writes into Buffer34 if the buffer is empty and its current frame has not been put into the buffer. Otherwise, task 3's writing fails, as stated in the challenge specification.

Task T4 Task T4 is modeled by a periodic PTA, and is essentially similar to task T3.

Deriving the latency for $n = 1$ Now, let us derive the latency for $n = 1$. As we have seen, in order to avoid exploring the exact configurations in the system, we target a single frame that is output from task 1 at $t = WCET_1$. The main idea is that, at $t = WCET_1$, the initial state must be *arbitrary*, i. e., encode all possible configurations that could happen in the system. However, such a model may be pessimistic for containing behaviors that cannot really happen in the system. Again, we aim to derive upper and lower bounds on end-to-end latency of an arbitrary frame.

After developing the model, we use IMITATOR to perform EF-synthesis of location T4end_ok. Then, IMITATOR hides (using existential quantification) all parameters except $E2E$, and then returns the following:

$$E2E \in [63, 145.008].$$

Quite interestingly, not hiding the other parameters gives a parametric interval, i. e., it is possible to know for which values of the parameters ($P3\_uncertain$ and $P4\_uncertain$) the best and worst cases occur.

Deriving the latency for $n = 3$ For the case of $n = 3$ for Buffer34, we can keep the same IMITATOR model, with the exception of the buffer modeling. We encode the buffer using extra variables. Details are given in [ALS15].

We obtain the following result:

$$E2E \in [63, 225.016].$$

Interestingly, our solution using IMITATOR was, to the best of our knowledge, the only one to compute these exact values—with one exception: a solution computed using simulation obtained 226 for $n = 3$ (rounded to the upper integer), which is correct, but could only be considered as a lower bound as it was computed by simulation (other runs could have been missed). Still, thanks to our result, we can confirm their solution is correct. This justifies my interest in studying the combination of simulation with parametric timed model checking.

6.4 Perspectives

Decidability There is still some gap between our decidability result (Theorem 6.2) and our undecidability results (Theorem 6.1) for PTaskAs. A promising way to improve the knowledge of decidability would be to show that L/U-parametric timed automata with bounded subtractions are decidable, which would allow in turn to extend our decidable subclass of PTaskA. Conversely, a likely candidate for undecidability is non-preemptive strategies without the schedulable-boundedness assumption.

In [Bér+13] parametric interrupt timed automata are proposed: this class inspired by PTAs is such that, at any time, at most one clock is active. This class allows a kind of preemption, and the reachability-emptiness problem is decidable. While it does not seem to be able to model general real-time systems, extending this class to model real-time systems while preserving decidability would be an interesting perspective.

Scalability While formal methods with timing parameters might not scale to verify the schedulability of very large systems with all details, we believe they can provide designers with first schedulability results on subparts of the system, or to derive timing bounds on abstractions of it.

Still, trying to achieve a better scalability is an important future work. Beyond promising methods not specifically dedicated to real-time systems (such as compositional parametric verification [Aşt+16; AL17b], or the algorithms and heuristics of Chapters 4 and 5), dedicated methods for achieving real scalability so as to compete with the scalability offered by analytical methods remains to be done. A first step towards this goal is the timed interfaces for real-time components we defined in [Lip+14]; further works (notably with Giuseppe Lipari and Sun Youcheng) could be based on this first step to improve scalability.

IMITATOR and real-time systems So far, IMITATOR takes as input networks of parametric timed automata extended with some convenient features such as variables, stopwatches or synchronization. Nevertheless, IMITATOR does not interface yet with common input formats for real-time systems. An ongoing work is the translation from a standard developed by Thales to the input format of IMITATOR so as to allow a smooth interface. Future works also include providing a better feedback (e. g., using graphics) to real-time systems experts.

Chapter 7

Conclusion

This thesis summarizes a selection of my contributions in parametric timed model checking. I studied the decidability and the expressiveness of the very expressive class of parametric timed automata, and several of its subclasses, and exhibited decidability results (Chapter 3). I then addressed the more practical problem to devise efficient algorithms for parameter synthesis (Chapter 4). I then devised algorithms dedicated to solving parametric problems for parametric timed automata (or parametric time Petri nets) extended with stopwatches, with action parameters or with probabilities (Chapter 5). Finally, I showed that parametric timed automata (and close formalisms) can be used to solve parametric schedulability analysis problems for real-time systems, with applications to actual industrial case studies (Chapter 6).

At the end of each chapter, I gave some research perspectives; I summarize the most important below.

Theory

Open subclasses L-PTAs and U-PTAs [BL09] are very open classes, in the sense that the only known decidability results come from the larger class of PTAs, and no undecidability result was known—with the exception of our recent result concerning TCTL-emptiness [ALR18]. To summarize, the EG-emptiness, AG-emptiness and AF-emptiness problems, as well as the language- and trace-preservation problems, are all undecidable for (general) L/U-PTAs, but remain open for L-PTAs and U-PTAs. Similarly, the EF-synthesis problem (shown intractable for L/U-PTAs in [JLR15] despite the decidability of the EF-emptiness problem) remains open for L- and U-PTAs, and would significantly increase the interest of these subclasses if it was shown to be computable.

Beyond timed automata Hybrid automata represent a generalization of timed automata, and become very powerful, with mostly undecidability results for the general class, while decidable subclasses were proposed (e. g., [Hen+98; Bri+13]), and several tools have been developed, notably SpaceEx [Fre+11]. Hybrid systems represent a challenging opportunity with many applications, notably in biology (e. g., [Sch+12; DD13; Roc+16; Lan+17], or more generally the entire "Hybrid Systems Biology" series). Adding timing parameters (that have natural applications, notably in biology) is a natural extension of hybrid systems. On the theoretical side, the work can be twofold: on the one hand, extending existing decidable subclasses of parametric timed automata with more general variables than clocks; a promising option is our class of reset-PTA that could be extended with initialization conditions [Hen+98]. And, on the other hand, extending existing decidable subclasses of hybrid systems with parameters; possible candidates are O-minimal hybrid systems [Bri+04], and hybrid automata with monotonic variables [Bri+13]. The ongoing PhD of Mathias Ramparison (co-supervised by Didier Lime) should address this direction of research.

From a more pragmatic point of view, some of the results of this manuscript could be extended to hybrid systems or combined with existing results for hybrid systems, such as combining concrete and symbolic executions, or studying various definitions of robustness in the hybrid setting (e. g., [FK13; AC15; AH15]).

Finally, the class of (parametric) polynomial interrupt automata [Bér+15] combines parameters with a generalization of the notion of clock (with some restriction though), and may be extended further.

Beyond timing parameters I mainly considered a unique type of parameters, i. e., *timing* parameters. On the one hand, *discrete* parameters in the sense of an unbounded number of (identical) processes, which represent a long line of work (e. g., [AJ03; ADM04; Abd+16] for timed extensions) could be extended with timing parameters. While the general class would be with no doubt undecidable, combining decidable subclasses of parametric timed automata (e. g., 1-clock, or L/U-PTA) with decidable subclasses of networks of processes (depending on the number of clocks, or on communication topologies) is a promising future work. This should be considered in the remainder of the ANR PACS project. Also note that other formalisms, not necessarily automata-based, such as regular model-checking [Bou+00], could also be extended to timing parameters.

On the other hand, the combination of (parametric) timed systems with probabilistic parameters (which were considered in e. g., [LMT04; LMT07; Ces+16; Su+16; Qua+16; DLP16]) would lead to powerful systems with challenging problems.

Controller synthesis Parameter synthesis consists in tuning a part of the model so that it satisfies its specification. A natural future work is controller synthesis, which consists in synthesizing the model itself. Combining controller synthesis and parameter synthesis is challenging, with potential industrial applications. A first approach relied in [JLR13] on the integer-parameter synthesis from [JLR15] and was able to compute the set of winning states for a parametric game automaton together with the (bounded) integer parameter valuations. Extending this work with rational-valued parameters (e. g., starting from [ALR15]) would be a first natural extension.

Applications

I have always been interested in the practical applications of theoretical research, and will continue to do so. The success of IMITATOR in the FMTV challenge (Section 6.3) led to a collaboration with Thales R&D. This collaboration will be a natural source of case studies, calling for new techniques, new heuristics, perhaps even new formalisms to solve practical problems in a reasonable time.

More generally, parametric timed formalisms could be used in the domain of cybersecurity (as showed a sample example in [And+16], and more in-depth algorithms for the verification of security protocols under uncertainty [Li+15; LSD16]).

My publications

[ABC14a] Étienne André, Mohamed Mahdi Benmoussa, and Christine Choppy. "Formalising Concurrent UML State Machines Using Coloured Petri Nets". In: *KSE*. Ed. by Viet-Ha Nguyen, Anh-Cuong Le, and Van-Nam Huynh. Vol. 326. Advances in Intelligent Systems and Computing. Hanoi, Vietnam: Springer, 2014, pp. 473–486. DOI: 10.1007/978-3-319-11680-8_38 (cit. on p. 9).

[ABC14b] Étienne André, Mohamed Mahdi Benmoussa, and Christine Choppy. "Translating UML State Machines to Coloured Petri Nets Using Acceleo: A Report". In: *ESSS*. Ed. by Yang Liu and Jun Pang. Vol. 150. Electronic Proceedings in Theoretical Computer Science. Singapore, 2014, pp. 1–7. DOI: 10.4204/EPTCS.150.1 (cit. on p. 9).

[ABC16] Étienne André, Mohamed Mahdi Benmoussa, and Christine Choppy. "Formalising concurrent UML state machines using coloured Petri nets". In: *Formal Aspects of Computing* 28.5 (2016), pp. 805–845. DOI: 10.1007/s00165-016-0388-9 (cit. on p. 9).

[ACE14] Étienne André, Camille Coti, and Sami Evangelista. "Distributed Behavioral Cartography of Timed Automata". In: *EuroMPI/ASIA*. Ed. by Jack Dongarra, Yutaka Ishikawa, and Hori Atsushi. Kyoto, Japan: ACM, 2014, pp. 109–114. DOI: 10.1145/2642769.2642784 (cit. on pp. 8, 58, 68).

[ACK12] Étienne André, Christine Choppy, and Kais Klai. "Formalizing non-concurrent UML state machines using colored Petri nets". In: *ACM SIGSOFT Software Engineering Notes* 37.4 (2012). Proceedings of the 5th International workshop UML and Formal Methods (UML&FM), pp. 1–8. DOI: 10.1145/2237796.2237819 (cit. on p. 9).

[ACN14] Étienne André, Christine Choppy, and Thierry Noulamo. "Modelling Timed Concurrent Systems Using Activity Diagram Patterns". In: *KSE*. Ed. by Viet-Ha Nguyen, Anh-Cuong Le, and Van-Nam Huynh. Vol. 326. Advances in Intelligent Systems and Computing. Hanoi, Vietnam: Springer, 2014, pp. 339–351. DOI: 10.1007/978-3-319-11680-8_27 (cit. on p. 9).

[ACN15] Étienne André, Camille Coti, and Hoang Gia Nguyen. "Enhanced Distributed Behavioral Cartography of Parametric Timed Automata". In: *ICFEM*. Ed. by Michael Butler, Sylvain Conchon, and Fatiha Zaïdi. Vol. 9407. Lecture Notes in Computer Science. Paris, France: Springer, 2015, pp. 319–335. ISBN: 978-3-319-25422-7. DOI: 10.1007/978-3-319-25423-4_21 (cit. on pp. 8, 58, 59, 68).

[ACR13] Étienne André, Christine Choppy, and Gianna Reggio. "Activity Diagrams Patterns for Modeling Business Processes". In: *SERA*. Ed. by Roger Y. Lee. Vol. 496. Studies in Computational Intelligence. Prague, Czech Republic: Springer, 2013, pp. 197–213. DOI: 10.1007/978-3-319-00948-3_13 (cit. on p. 9).

[ACR17] Étienne André, Thomas Chatain, and César Rodríguez. “Preserving Partial Order Runs in Parametric Time Petri Nets”. In: *Transactions on Embedded Computing Systems* 16.2 (2017), 43:1–43:26. DOI: 10.1145/3012283 (cit. on pp. 9, 71, 73, 75–78).

[AD16] Étienne André and Benoit Delahaye. “Consistency in Parametric Interval Probabilistic Timed Automata”. In: *TIME*. Ed. by Curtis E. Dyreson, Michael R. Hansen, and Luke Hunsberger. Kongens Lyngby, Denmark: IEEE Computer Society, 2016, pp. 110–119. DOI: 10.1109/TIME.2016.19 (cit. on pp. 9, 82–85).

[AF10] Étienne André and Laurent Fribourg. “Behavioral Cartography of Timed Automata”. In: *RP*. Ed. by Antonín Kučera and Igor Potapov. Vol. 6227. Lecture Notes in Computer Science. Brno, Czech Republic: Springer, 2010, pp. 76–90. DOI: 10.1007/978-3-642-15349-5_5 (cit. on pp. 29, 56, 88).

[AFS13a] Étienne André, Laurent Fribourg, and Romain Soulat. “Merge and Conquer: State Merging in Parametric Timed Automata”. In: *ATVA*. Ed. by Dang-Van Hung and Mizuhito Ogawa. Vol. 8172. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, 2013, pp. 381–396. DOI: 10.1007/978-3-319-02444-8_27 (cit. on pp. 8, 48, 49).

[AFS13b] Étienne André, Laurent Fribourg, and Jeremy Sproston. “An Extension of the Inverse Method to Probabilistic Timed Automata”. In: *Formal Methods in System Design* 2 (2013), pp. 119–145. DOI: 10.1007/s10703-012-0169-x (cit. on p. 82).

[AL17a] Étienne André and Didier Lime. “Liveness in L/U-Parametric Timed Automata”. In: *ACSD*. Ed. by Alex Legay and Klaus Schneider. Zaragoza, Spain: IEEE, 2017, pp. 9–18. DOI: 10.1109/ACSD.2017.19 (cit. on pp. 8, 18, 32, 36–40, 44, 63, 85).

[AL17b] Étienne André and Shang-Wei Lin. “Learning-based compositional parameter synthesis for event-recording automata”. In: *FORTE*. Ed. by Ahmed Bouajjani and Silva Alexandra. Vol. 10321. Lecture Notes in Computer Science. Best FORTE paper award and best DisCoTec paper award. Neuchâtel, Switzerland: Springer, 2017, pp. 17–32. DOI: 10.1007/978-3-319-60225-7_2 (cit. on pp. 8, 64, 67, 69, 97).

[AL18] Étienne André and Shang-Wei Lin. “The language preservation problem is undecidable for parametric event-recording automata”. In: *Information Processing Letters* 136 (2018), pp. 17–20. DOI: 10.1016/j.ipl.2018.03.013 (cit. on pp. 8, 65).

[ALM18] Étienne André, Didier Lime, and Nicolas Markey. *Language Preservation Problems in Parametric Timed Automata (extended version)*. arXiv:1807.07091. arXiv, 2018 (cit. on pp. 42, 43).

[ALR15] Étienne André, Didier Lime, and Olivier H. Roux. “Integer-Complete Synthesis for Bounded Parametric Timed Automata”. In: *RP*. Ed. by Mikołaj Bojańczyk, Sławomir Lasota, and Igor Potapov. Vol. 9328. Lecture Notes in Computer Science. Warsaw, Poland: Springer, 2015, pp. 7–19. DOI: 10.1007/978-3-319-24537-9_2 (cit. on pp. 53–55, 99).

[ALR16a] Étienne André, Didier Lime, and Olivier H. Roux. “Decision Problems for Parametric Timed Automata”. In: *ICFEM*. Ed. by Kazuhiro Ogata, Mark Lawford, and Shaoying Liu. Vol. 10009. Lecture Notes in Computer Science. Tokyo, Japan: Springer, 2016, pp. 400–416. DOI: 10.1007/978-3-319-47846-3_25 (cit. on pp. 8, 17, 18, 27, 28, 30, 32, 34–37, 39, 44).

[ALR16b] Étienne André, Didier Lime, and Olivier H. Roux. “On the Expressiveness of Parametric Timed Automata”. In: *FORMATS*. Ed. by Martin Fränzle and Nicolas Markey. Vol. 9984. Lecture Notes in Computer Science. Québec, Canada: Springer, 2016, pp. 19–34. DOI: 10.1007/978-3-319-44878-7_2 (cit. on pp. 8, 27, 29–32, 46).

[ALR18] Étienne André, Didier Lime, and Mathias Ramparison. “TCTL model checking lower/upper-bound parametric timed automata without invariants”. In: *FORMATS*. Ed. by David N. Jansen and Pavithra Prabhakar. Vol. 11022. Lecture Notes in Computer Science. To appear. Beijing, China: Springer, 2018, pp. 1–17. DOI: 10.1007/978-3-030-00151-3_3 (cit. on pp. 8, 32, 38, 44, 98).

[ALS15] Étienne André, Giuseppe Lipari, and Youcheng Sun. “Verification of Two Real-Time Systems Using Parametric Timed Automata”. In: *WATERS*. Ed. by Sophie Quinton and Tullio Vardanega. Lund, Sweden, 2015 (cit. on pp. 9, 95, 97).

[AM15] Étienne André and Nicolas Markey. “Language Preservation Problems in Parametric Timed Automata”. In: *FORMATS*. Ed. by Sriram Sankaranarayanan and Enrico Vicario. Vol. 9268. Lecture Notes in Computer Science. Madrid, Spain: Springer, 2015, pp. 27–43. DOI: 10.1007/978-3-319-22975-1_3 (cit. on pp. 8, 14, 20, 21, 29, 32, 39, 42–44, 65, 79, 85).

[And+09a] Étienne André, Thomas Chatain, Olivier De Smet, Laurent Fribourg, and Silvain Ruel. “Synthèse de contraintes temporisées pour une architecture d’automatisation en réseau”. In: *MSR*. Ed. by Didier Lime and Olivier H. Roux. Vol. 43. Journal Européen des Systèmes Automatisés 7-9. Nantes, France: Hermès, 2009, pp. 1049–1064 (cit. on p. 42).

[And+09b] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. “An Inverse Method for Parametric Timed Automata”. In: *International Journal of Foundations of Computer Science* 20.5 (2009), pp. 819–836. DOI: 10.1142/S0129054109006905 (cit. on pp. 14, 15, 20, 21, 48, 70, 79).

[And+12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. “IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems”. In: *FM*. Ed. by Dimitra Giannakopoulou and Dominique Méry. Vol. 7436. Lecture Notes in Computer Science. Paris, France: Springer, 2012, pp. 33–36. DOI: 10.1007/978-3-642-32759-9_6 (cit. on pp. 9, 47, 67).

[And+13] Étienne André, Yang Liu, Jun Sun, Jin Song Dong, and Shang-Wei Lin. “PSyHCoS: Parameter Synthesis for Hierarchical Concurrent Real-Time Systems”. In: *CAV*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Saint Petersburg, Russia: Springer, 2013, pp. 984–989. DOI: 10.1007/978-3-642-39799-8_70 (cit. on p. 9).

[And+14] Étienne André, Yang Liu, Jun Sun, and Jin Song Dong. “Parameter Synthesis for Hierarchical Concurrent Real-Time Systems”. In: *Real-Time Systems* 50.5-6 (2014), pp. 620–679. DOI: 10.1007/s11241-014-9208-6 (cit. on pp. 9, 49).

[And+15] Étienne André, Giuseppe Lipari, Hoang Gia Nguyen, and Youcheng Sun. “Reachability Preservation Based Parameter Synthesis for Timed Automata”. In: *NFM*. Ed. by Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi. Vol. 9058. Lecture Notes in Computer Science. Pasadena, CA, USA: Springer, 2015, pp. 50–65. DOI: 10.1007/978-3-319-17524-9_5 (cit. on pp. 8, 61–63).

[And+16] Étienne André, Michał Knapik, Wojciech Penczek, and Laure Petrucci. “Controlling Actions and Time in Parametric Timed Automata”. In: *ACSD*. Ed. by Jörg Desel and Alex Yakovlev. Toruń, Poland: IEEE Computer Society, 2016, pp. 45–54. DOI: 10.1109/ACSD.2016.20 (cit. on pp. 9, 81, 82, 99).

[And+17a] Étienne André, Michał Knapik, Wojciech Jamroga Wojciech Penczek, and Laure Petrucci. “Timed ATL: Forget Memory, Just Count”. In: *AAMAS*. Ed. by Kate Larson, Michael Winikoff, Sanmay Das, and Edmund Durfee. São Paulo, Brazil: ACM, 2017, pp. 1460–1462 (cit. on p. 10).

[And+17b] Étienne André, Hoang Gia Nguyen, Laure Petrucci, and Jun Sun. “Parametric model checking timed automata under non-Zenoness assumption”. In: *NFM*. Ed. by Clark Barrett and Temesghen Kahsai. Vol. 10227. Lecture Notes in Computer Science. Moffett Field, CA, USA: Springer, 2017, pp. 35–51. DOI: 10.1007/978-3-319-57288-8_3 (cit. on pp. 9, 78–80).

[And13a] Étienne André. “Dynamic Clock Elimination in Parametric Timed Automata”. In: *FSFMA*. Ed. by Christine Choppy and Jun Sun. Vol. 31. OpenAccess Series in Informatics (OASIcs). Singapore: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2013, pp. 18–31. DOI: 10.4230/OASIcs.FSFMA.2013.18 (cit. on pp. 8, 50).

[And13b] Étienne André. “Observer Patterns for Real-Time Systems”. In: *ICECCS*. Ed. by Yang Liu and Andrew Martin. Singapore: IEEE Computer Society, 2013, pp. 125–134. DOI: 10.1109/ICECCS.2013.26 (cit. on p. 9).

[And16] Étienne André. “Parametric Deadlock-Freeness Checking Timed Automata”. In: *ICTAC*. Ed. by Augusto Cesar Alves Sampaio and Farn Wang. Vol. 9965. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, 2016, pp. 469–478. DOI: 10.1007/978-3-319-46750-4_27 (cit. on pp. 8, 37, 40–42, 44).

[And17] Étienne André. “A unified formalism for monoprocessor schedulability analysis under uncertainty”. In: *FMICS-AVoCS*. Ed. by Ana Cavalcanti, Laure Petrucci, and Cristina Seceleanu. Vol. 10471. Lecture Notes in Computer Science. Best paper award. Torino, Italy: Springer, 2017, pp. 100–115. DOI: 10.1007/978-3-319-67113-0_7 (cit. on pp. 9, 91–94).

[And18] Étienne André. “What’s decidable about parametric timed automata?” In: *International Journal on Software Tools for Technology Transfer* (2018). To appear. DOI: 10.1007/s10009-017-0467-0 (cit. on pp. 8, 17, 22).

[ANP17] Étienne André, Hoang Gia Nguyen, and Laure Petrucci. “Efficient parameter synthesis using optimized state exploration strategies”. In: *ICECCS*. Ed. by Zhenjiang Hu and Guangdong Bai. Fukuoka, Japan: IEEE, 2017, pp. 1–10. DOI: 10.1109/ICECCS.2017.28 (cit. on p. 10).

[AP15] Étienne André and Laure Petrucci. “Unifying Patterns for Modelling Timed Relationships in Systems and Properties”. In: *PNSE*. Ed. by Daniel Moldt, Heiko Rölke, and Harald Störrle. Vol. 1372. Brussels, Belgium: CEUR-WS, 2015, pp. 25–40 (cit. on p. 9).

[APP13] Étienne André, Laure Petrucci, and Giuseppe Pellegrino. “Precise Robustness Analysis of Time Petri Nets with Inhibitor Arcs”. In: *FORMATS*. Ed. by Víctor Braberman and Laurent Fribourg. Vol. 8053. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer, 2013, pp. 1–15. DOI: 10.1007/978-3-642-40229-6_1 (cit. on pp. 9, 72–75).

[AS11] Étienne André and Romain Soulat. “Synthesis of Timing Parameters Satisfying Safety Properties”. In: *RP*. Ed. by Giorgio Delzanno and Igor Potapov. Vol. 6945. Lecture Notes in Computer Science. Genova, Italy: Springer, 2011, pp. 31–44. DOI: 10.1007/978-3-642-24288-5_5 (cit. on pp. 9, 60, 70–72).

[AS13] Étienne André and Romain Soulat. *The Inverse Method*. FOCUS Series in Computer Engineering and Information Technology. 176 pages. ISTE Ltd and John Wiley & Sons Inc., 2013. ISBN: 9781848214477 (cit. on pp. 15, 56, 70).

[Li+17] Jiaying Li, Jun Sun, Bo Gao, and Étienne André. “Classification based Parameter Synthesis for Parametric Timed Automata”. In: *ICFEM*. Ed. by Zhenhua Duan and Luke Ong. Vol. 10610. Lecture Notes in Computer Science. Xi’An, China: Springer, 2017, pp. 243–261. DOI: 10.1007/978-3-319-68690-5_15 (cit. on pp. 10, 69).

[Lin+11] Shang-Wei Lin, Étienne André, Jin Song Dong, Jun Sun, and Yang Liu. "An Efficient Algorithm for Learning Event-Recording Automata". In: *ATVA*. Ed. by Tevfik Bultan and Pao-Ann Hsiung. Vol. 6996. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, 2011, pp. 463–472. DOI: 10.1007/978-3-642-24372-1_35 (cit. on p. 65).

[Lin+12] Shang-Wei Lin, Yang Liu, Jun Sun, Jin Song Dong, and Étienne André. "Automatic Compositional Verification of Timed Systems". In: *FM*. Ed. by Dimitra Giannakopoulou and Dominique Méry. Vol. 7436. Lecture Notes in Computer Science. Paris, France: Springer, 2012, pp. 272–276. DOI: 10.1007/978-3-642-32759-9_24 (cit. on pp. 66, 67).

[Lin+14] Shang-Wei Lin, Étienne André, Yang Liu, Jun Sun, and Jin Song Dong. "Learning Assumptions for Compositional Verification of Timed Systems". In: *Transactions on Software Engineering* 40.2 (2014), pp. 137–153. DOI: 10.1109/TSE.2013.57 (cit. on pp. 8, 66).

[Lip+14] Giuseppe Lipari, Youcheng Sun, Étienne André, and Laurent Fribourg. "Toward Parametric Timed Interfaces for Real-Time Components". In: *SynCoP*. Ed. by Étienne Andre and Goran Frehse. Vol. 145. Electronic Proceedings in Theoretical Computer Science. Grenoble, France, 2014, pp. 49–64. DOI: 10.4204/EPTCS.145.6 (cit. on p. 97).

[Sun+13a] Jun Sun, Yang Liu, Jin Song Dong, Yan Liu, Ling Shi, and Étienne André. "Modeling and Verifying Hierarchical Real-time Systems using Stateful Timed CSP". In: *ACM Transactions on Software Engineering and Methodology* 22.1 (2013), pp. 3.1–3.29. DOI: 10.1145/2430536.2430537 (cit. on pp. 9, 49).

[Sun+13b] Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. "Parametric Schedulability Analysis of Fixed Priority Real-Time Distributed Systems". In: *FSTCS*. Ed. by Cyrille Artho and Peter Ölveczky. Vol. 419. Communications in Computer and Information Science. Auckland, New Zealand: Springer, 2013, pp. 212–228. DOI: 10.1007/978-3-319-05416-2_14 (cit. on pp. 9, 88–90).

[Tan+13] Tian Huat Tan, Étienne André, Jun Sun, Yang Liu, Jin Song Dong, and Manman Chen. "Dynamic Synthesis of Local Time Requirement for Service Composition". In: *ICSE*. Ed. by Betty H.C. Cheng and Klaus Pohl. San Francisco, USA: IEEE, 2013, pp. 542–551. DOI: 10.1109/ICSE.2013.6606600 (cit. on p. 10).

[Tan+14] Tian Huat Tan, Manman Chen, Étienne André, Jun Sun, Yang Liu, and Jin Song Dong. "Automated Runtime Recovery for QoS-based Service Composition". In: *WWW*. Ed. by Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel. Seoul, Korea: ACM SIG Proceedings, 2014, pp. 563–574. ISBN: 978-1-4503-2744-2. DOI: 10.1145/2566486.2568048 (cit. on p. 10).

[Tan+16] Tian Huat Tan, Manman Chen, Jun Sun, Yang Liu, Étienne André, Jin Song Dong, and Yinxing Xue. "Optimizing Selection of Competing Services with Probabilistic Hierarchical Refinement". In: *ICSE*. Ed. by Willem Visser and Laurie Williams. Austin, Texas, USA: ACM, 2016, pp. 85–95. ISBN: 978-1-4503-3900-1. DOI: http://doi.acm.org/10.1145/2884781.2884861 (cit. on p. 10).

References

[Abd+16] Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. "Parameterized verification of time-sensitive models of ad hoc network protocols". In: *Theoretical Computer Science* 612 (2016), pp. 1–22. DOI: 10.1016/j.tcs.2015.07.048 (cit. on p. 99).

[AC15] Ayman Aljarbouh and Benoît Caillaud. "Robust Simulation for Hybrid Systems: Chattering Bath Avoidance". In: *SIMS*. Linköping, Sweden, 2015, pp. 175–185. DOI: 10.3384/ecp15119175 (cit. on pp. 78, 99).

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. "Model-Checking in Dense Real-Time". In: *Information and Computation* 104.1 (1993), pp. 2–34 (cit. on p. 23).

[AD94] Rajeev Alur and David L. Dill. "A theory of timed automata". In: *Theoretical Computer Science* 126.2 (1994), pp. 183–235. ISSN: 0304-3975 (cit. on pp. 6, 12, 17, 23, 26, 30, 37, 45, 65).

[ADM04] Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. "Multi-Clock Timed Networks". In: *LICS*. Turku, Finland: IEEE Computer Society, 2004, pp. 345–354. DOI: 10.1109/LICS.2004.1319629 (cit. on p. 99).

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. "Event-Clock Automata: A Determinizable Class of Timed Automata". In: *Theoretical Computer Science* 211.1-2 (1999), pp. 253–273. DOI: 10.1016/S0304-3975(97)00173-4 (cit. on pp. 8, 63, 65).

[AH15] Takumi Akazaki and Ichiro Hasuo. "Time Robustness in MTL and Expressivity in Hybrid System Falsification". In: *CAV, part II*. Ed. by Daniel Kroening and Corina S. Pasareanu. Vol. 9207. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, 2015, pp. 356–374. DOI: 10.1007/978-3-319-21668-3_21 (cit. on pp. 69, 99).

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. "Parametric real-time reasoning". In: *STOC*. Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. San Diego, California, United States: ACM, 1993, pp. 592–601. ISBN: 0-89791-591-7 (cit. on pp. 7, 12, 17, 19, 22–25, 30, 44, 81, 92).

[AJ03] Parosh Aziz Abdulla and Bengt Jonsson. "Model checking of systems with many identical timed processes". In: *Theoretical Computer Science* 290.1 (2003), pp. 241–264. DOI: 10.1016/S0304-3975(01)00330-9 (cit. on p. 99).

[Aks+16] S. Akshay, Loïc Hélouët, Claude Jard, and Pierre-Alain Reynier. "Robustness of Time Petri Nets under Guard Enlargement". In: *Fundamenta Informaticae* 143.3-4 (2016), pp. 207–234. DOI: 10.3233/FI-2016-1312 (cit. on pp. 43, 44, 73, 75, 85).

[Alj+16] Ayman Aljarbouh, Yingfu Zeng, Adam Duracz, Benoît Caillaud, and Walid Taha. "Chattering-Free Simulation for Hybrid Dynamical Systems Semantics and Prototype Implementation". In: *CSE, EUC, DCABES*. Paris, France: IEEE Computer Society, 2016, pp. 412–422. DOI: 10.1109/CSE-EUC-DCABES.2016.217 (cit. on p. 78).

[AM01] Yasmina Abdeddaïm and Oded Maler. "Job-Shop Scheduling Using Timed Automata". In: *CAV*. Ed. by Gérard Berry, Hubert Comon, and Alain Finkel. Vol. 2102. Lecture Notes in Computer Science. Paris, France: Springer, 2001, pp. 478–492. ISBN: 3-540-42345-1. DOI: 10.1007/3-540-44585-4_46 (cit. on p. 88).

[AM02] Yasmina Adbeddaïm and Oded Maler. "Preemptive Job-Shop Scheduling using Stopwatch Automata". In: *TACAS*. Ed. by Joost-Pieter Katoen and Perdita Stevens. Vol. 2280. Lecture Notes in Computer Science. Grenoble, France: Springer-Verlag, 2002, pp. 113–126 (cit. on p. 88).

[Ang87] Dana Angluin. "Learning Regular Sets from Queries and Counterexamples". In: *Information and Computation* 75.2 (1987), pp. 87–106. DOI: 10.1016/0890-5401(87)90052-6 (cit. on p. 65).

[Aşt+16] Lăcrămioara Aştefănoaei, Saddek Bensalem, Marius Bozga, Chih-Hong Cheng, and Harald Ruess. "Compositional Parameter Synthesis". In: *FM*. Ed. by John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou. Vol. 9995. Lecture Notes in Computer Science. 2016, pp. 60–68. DOI: 10.1007/978-3-319-48989-6_4 (cit. on pp. 7, 69, 97).

[Aud91] Neil C. Audsley. *Optimal Priority Assignment And Feasibility Of Static Priority Tasks With Arbitrary Start Times*. Tech. rep. YCS 164. Dept computer science, University of York, 1991 (cit. on p. 87).

[Bak91] Theodore P. Baker. "Stack-based Scheduling of Realtime Processes". In: *Real-Time Systems* 3.1 (1991), pp. 67–99. DOI: 10.1007/BF00365393 (cit. on p. 88).

[BBM06] Ramzi Ben Salah, Marius Bozga, and Oded Maler. "On Interleaving in Timed Automata". In: *CONCUR*. Ed. by Christel Baier and Holger Hermanns. Vol. 4137. Lecture Notes in Computer Science. Bonn, Germany: Springer, 2006, pp. 465–476. ISBN: 3-540-37376-4. DOI: 10.1007/11817949_31 (cit. on p. 48).

[Beh+06] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. "Lower and upper bounds in zone-based abstractions of timed automata". In: *International Journal on Software Tools for Technology Transfer* 8.3 (2006), pp. 204–215. DOI: 10.1007/s10009-005-0190-0 (cit. on pp. 52, 53).

[Ben+15] Nikola Beneš, Peter Bezděk, Kim G. Larsen, and Jiří Srba. "Language Emptiness of Continuous-Time Parametric Timed Automata". In: *ICALP, Part II*. Vol. 9135. Lecture Notes in Computer Science. Springer, 2015, pp. 69–81. DOI: 10.1007/978-3-662-47666-6_6 (cit. on pp. 7, 24, 25, 29, 30, 37, 38, 42, 85).

[Bér+13] Béatrice Bérard, Serge Haddad, Aleksandra Jovanovic, and Didier Lime. "Parametric Interrupt Timed Automata". In: *RP*. Ed. by Parosh Aziz Abdulla and Igor Potapov. Vol. 8169. Lecture Notes in Computer Science. Springer, 2013, pp. 59–69. DOI: 10.1007/978-3-642-41036-9_7 (cit. on p. 97).

[Bér+15] Béatrice Bérard, Serge Haddad, Claudine Picaronny, Mohab Safey El Din, and Mathieu Sassolas. "Polynomial Interrupt Timed Automata". In: *RP*. Ed. by Mikolaj Bojanczyk, Slawomir Lasota, and Igor Potapov. Vol. 9328. Lecture Notes in Computer Science. Warsaw, Poland: Springer, 2015, pp. 20–32. DOI: 10.1007/978-3-319-24537-9_3 (cit. on p. 99).

[BG06] Howard Bowman and Rodolfo Gómez. "How to stop time stopping". In: *Formal Aspects of Computing* 18.4 (2006), pp. 459–493. DOI: 10.1007/s00165-006-0010-7 (cit. on p. 78).

[BHZ08] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. “The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems”. In: *Science of Computer Programming* 72.1–2 (2008), pp. 3–21. DOI: 10.1016/j.scico.2007.08.001 (cit. on pp. 48, 49, 55, 78).

[BL09] Laura Bozzelli and Salvatore La Torre. “Decision problems for lower/upper bound parametric timed automata”. In: *Formal Methods in System Design* 35.2 (2009), pp. 121–151. DOI: 10.1007/s10703-009-0074-0 (cit. on pp. 7, 16, 26, 35, 38, 98).

[BMS11] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. “Robust Model-Checking of Timed Automata via Pumping in Channel Machines”. In: *FORMATS*. Ed. by Uli Fahrenberg and Stavros Tripakis. Vol. 6919. Lecture Notes in Computer Science. Aalborg, Denmark: Springer, 2011, pp. 97–112. DOI: 10.1007/978-3-642-24310-3_8 (cit. on p. 44).

[BMS13] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. “Robustness in timed automata”. In: *RP*. Ed. by Parosh Aziz Abdulla and Igor Potapov. Vol. 8169. Lecture Notes in Computer Science. Invited paper. Uppsala, Sweden: Springer, 2013, pp. 1–18. DOI: 10.1007/978-3-642-41036-9_1 (cit. on pp. 7, 51).

[BO14] Daniel Bundala and Joël Ouaknine. “Advances in Parametric Real-Time Reasoning”. In: *MFCS, Part I*. Ed. by Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik. Vol. 8634. Lecture Notes in Computer Science. Budapest, Hungary: Springer, 2014, pp. 123–134. ISBN: 978-3-662-44521-1. DOI: 10.1007/978-3-662-44522-8 (cit. on pp. 7, 24, 25).

[Bou+00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. “Regular Model Checking”. In: *CAV*. Ed. by E. Allen Emerson and A. Prasad Sistla. Vol. 1855. Lecture Notes in Computer Science. Chicago, IL, USA: Springer, 2000, pp. 403–418. DOI: 10.1007/10722167_31 (cit. on p. 99).

[BRH90] Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell. “Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor”. In: *Real-Time Systems* 2.4 (1990), pp. 301–324. DOI: 10.1007/BF01995675 (cit. on p. 87).

[Bri+04] Thomas Brihaye, Christian Michaux, Cédric Rivière, and Christophe Troestler. “On O-Minimal Hybrid Systems”. In: *HSCC*. Ed. by Rajeev Alur and George J. Pappas. Vol. 2993. Lecture Notes in Computer Science. Springer, 2004, pp. 219–233. DOI: 10.1007/978-3-540-24743-2_15 (cit. on p. 98).

[Bri+13] Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joël Ouaknine, Jean-François Raskin, and James Worrell. “Time-Bounded Reachability for Monotonic Hybrid Automata: Complexity and Fixed Points”. In: *ATVA*. Ed. by Dang Van Hung and Mizuhito Ogawa. Vol. 8172. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, 2013, pp. 55–70. DOI: 10.1007/978-3-319-02444-8_6 (cit. on pp. 23, 43, 85, 98).

[Bul+11] Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Marius Mikucionis, and Axel Legay. “Distributed Parametric and Statistical Model Checking”. In: *PDMC*. Ed. by Jiri Barnat and Keijo Heljanko. Vol. 72. EPTCS. Snowbird, Utah, USA, 2011, pp. 30–42. DOI: 10.4204/EPTCS.72.4 (cit. on p. 47).

[BY03] Johan Bengtsson and Wang Yi. “Timed Automata: Semantics, Algorithms and Tools”. In: *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*. Ed. by Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg. Vol. 3098. Lecture Notes in Computer Science. Eichstätt, Germany: Springer, 2003, pp. 87–124. DOI: 10.1007/978-3-540-27755-2_3 (cit. on pp. 48, 49).

[CC04] Robert Clarisó and Jordi Cortadella. “Verification of timed circuits with symbolic delays”. In: *ASP-DAC*. Ed. by Masaharu Imai. Yokohama, Japan: IEEE Computer Society, 2004, pp. 628–633. ISBN: 0-7803-8175-0. DOI: 10.1109/ASPDAC.2004.208 (cit. on p. 42).

[CC05] Robert Clarisó and Jordi Cortadella. “Verification of Concurrent Systems with Parametric Delays Using Octahedra”. In: *ACSD*. Saint-Malo, France: IEEE Computer Society, 2005, pp. 122–131. ISBN: 0-7695-2363-3. DOI: 10.1109/ACSD.2005.34 (cit. on p. 42).

[Ces+16] Milan Ceska, Petr Pilar, Nicola Paoletti, Lubos Brim, and Marta Z. Kwiatkowska. “PRISM-PSY: Precise GPU-Accelerated Parameter Synthesis for Stochastic Systems”. In: *TACAS*. Ed. by Marsha Chechik and Jean-François Raskin. Vol. 9636. Lecture Notes in Computer Science. Eindhoven, The Netherlands: Springer, 2016, pp. 367–384. DOI: 10.1007/978-3-662-49674-9_21 (cit. on pp. 47, 99).

[Cha+08] Najla Chamseddine, Marie Duflot, Laurent Fribourg, Claudine Picaronny, and Jeremy Sproston. “Computing Expected Absorption Times for Parametric Determinate Probabilistic Timed Automata”. In: *QEST*. Saint-Malo, France: IEEE Computer Society, 2008, pp. 254–263. DOI: 10.1109/QEST.2008.34 (cit. on p. 82).

[Che+09] Rémy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. “Timed Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata”. In: *Formal Methods in System Design* 34.1 (2009), pp. 59–81. DOI: 10.1007/s10703-008-0061-x (cit. on p. 7).

[CL00] Franck Cassez and Kim Guldstrand Larsen. “The Impressive Power of Stopwatches”. In: *CONCUR*. Ed. by Catuscia Palamidessi. Vol. 1877. Lecture Notes in Computer Science. Springer, 2000, pp. 138–152. DOI: 10.1007/3-540-44618-4_12 (cit. on pp. 74, 88).

[CL90] Min-Ih Chen and Kwei-Jay Lin. “Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time”. In: *Real-Time Systems* 2.4 (1990), pp. 325–346. DOI: 10.1007/BF01995676 (cit. on p. 88).

[CPR08] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. “Symbolic Computation of Schedulability Regions Using Parametric Timed Automata”. In: *RTSS*. Barcelona, Spain: IEEE Computer Society, 2008, pp. 80–89. ISBN: 978-0-7695-3477-0. DOI: 10.1109/RTSS.2008.36 (cit. on pp. 70, 88).

[CS01] Aurore Collomb–Annichini and Mihaela Sighireanu. “Parameterized Reachability Analysis of the IEEE 1394 Root Contention Protocol using TReX”. In: *RT-TOOLS*. Alborg, Danemark, 2001 (cit. on p. 42).

[CSB90] Houssine Chetto, Maryline Silly, and T. Bouchentouf. “Dynamic Scheduling of Real-Time Tasks under Precedence Constraints”. In: *Real-Time Systems* 2.3 (1990), pp. 181–194. DOI: 10.1007/BF00365326 (cit. on p. 88).

[DAr+97] Pedro R. D’Argenio, Joost-Pieter Katoen, Theo C. Ruys, and Jan Tretmans. “The Bounded Retransmission Protocol Must Be on Time!” In: *TACAS*. Ed. by Ed Brinksma. Vol. 1217. Lecture Notes in Computer Science. Enschede, The Netherlands: Springer, 1997, pp. 416–431. ISBN: 3-540-62790-1. DOI: 10.1007/BFb0035403 (cit. on p. 81).

[Dav+07] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. “Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised”. In: *Real-Time Systems* 35.3 (2007), pp. 239–272. DOI: 10.1007/s11241-007-9012-7 (cit. on p. 88).

[Dav05] Alexandre David. “Merging DBMs Efficiently”. In: *NWPT*. DIKU, University of Copenhagen, 2005, pp. 54–56 (cit. on p. 48).

[Dav06] Alexandre David. *Uppaal DBM Library Programmer's Reference*. http://people.cs.aau.dk/~adavid/UDBM/manual-061023.pdf. 2006 (cit. on p. 48).

[DD13] Thao Dang and Tommaso Dreossi. "Falsifying Oscillation Properties of Parametric Biological Models". In: *HSB*. Ed. by Thao Dang and Carla Piazza. Vol. 125. EPTCS. Taormina, Italy, 2013, pp. 53–67. DOI: 10.4204/EPTCS.125.4 (cit. on p. 98).

[De +08] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. "Robust safety of timed automata". In: *Formal Methods in System Design* 33.1-3 (2008), pp. 45–84. DOI: 10.1007/s10703-008-0056-7 (cit. on p. 51).

[Del+12] Benoît Delahaye, Kim Gulstrand Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. "Consistency and refinement for Interval Markov Chains". In: *Journal of Logic and Algebraic Programming* 81.3 (2012), pp. 209–226. DOI: 10.1016/j.jlap.2011.10.003 (cit. on p. 82).

[Del15] Benoît Delahaye. "Consistency for Parametric Interval Markov Chains". In: *SynCoP*. Ed. by Étienne André and Goran Frehse. Vol. 44. OASICS. London, United Kingdom: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015, pp. 17–32. DOI: 10.4230/OASIcs.SynCoP.2015.17 (cit. on p. 84).

[DLP16] Benoît Delahaye, Didier Lime, and Laure Petrucci. "Parameter Synthesis for Parametric Interval Markov Chains". In: *VMCAI*. Ed. by Barbara Jobstmann and K. Rustan M. Leino. Vol. 9583. Lecture Notes in Computer Science. St. Petersburg, FL, USA: Springer, 2016, pp. 372–390. DOI: 10.1007/978-3-662-49122-5_18 (cit. on pp. 83, 99).

[Doy07] Laurent Doyen. "Robust Parametric Reachability for Timed Automata". In: *Information Processing Letters* 102.5 (2007), pp. 208–213. DOI: 10.1016/j.ipl.2006.11.018 (cit. on pp. 24, 25).

[DY96] Conrado Daws and Sergio Yovine. "Reducing the number of clock variables of timed automata". In: *RTSS*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 73–81. ISBN: 0-8186-7689-2. DOI: 10.1109/REAL.1996.563702 (cit. on p. 49).

[Eva+12] Sami Evangelista, Alfons Laarman, Laure Petrucci, and Jaco Van de Pol. "Improved Multi-Core Nested Depth-First Search". In: *ATVA*. Ed. by Supratik Chakraborty and Madhavan Mukund. Vol. 7561. Lecture Notes in Computer Science. Thiruvananthapuram, India: Springer, 2012, pp. 269–283. ISBN: 978-3-642-33385-9. DOI: 10.1007/978-3-642-33386-6_22 (cit. on p. 47).

[Fer+07] Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. "Task automata: Schedulability, decidability and undecidability". In: *Information and Computation* 205.8 (2007), pp. 1149–1172. DOI: 10.1016/j.ic.2007.01.009 (cit. on pp. 9, 91–93).

[FJ13] Léa Fanchon and Florent Jacquemard. "Formal Timing Analysis Of Mixed Music Scores". In: *ICMC*. Perth, Australia: Michigan Publishing, 2013 (cit. on p. 7).

[FK13] Laurent Fribourg and Ulrich Kühne. "Parametric Verification and Test Coverage for Hybrid Automata Using the Inverse Method". In: *International Journal of Foundations of Computer Science* 24.2 (2013), pp. 233–249 (cit. on p. 99).

[For+10] Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. "Scheduling Dependent Periodic Tasks without Synchronization Mechanisms". In: *RTAS*. Ed. by Marco Caccamo. Stockholm, Sweden: IEEE Computer Society, 2010, pp. 301–310. DOI: 10.1109/RTAS.2010.26 (cit. on p. 88).

[For+11] Julien Forget, Emmanuel Grolleau, Claire Pagetti, and Pascal Richard. “Dynamic priority scheduling of periodic tasks with extended precedences”. In: *ETFA*. Ed. by Zoubir Mammeri. Toulouse, France: IEEE, 2011, pp. 1–8. DOI: 10.1109/ETFA.2011.6059015 (cit. on p. 88).

[Fre+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. “SpaceEx: Scalable Verification of Hybrid Systems”. In: *CAV*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Snowbird, UT, USA: Springer, 2011, pp. 379–395. DOI: 10.1007/978-3-642-22110-1_30 (cit. on p. 98).

[Fri+12] Laurent Fribourg, David Lesens, Pierre Moro, and Romain Soulat. “Robustness Analysis for Scheduling Problems using the Inverse Method”. In: *TIME*. Ed. by Mark Reynolds, Paolo Terenziani, and Ben Moszkowski. Leicester, UK: IEEE Computer Society Press, 2012, pp. 73–80. DOI: 10.1109/TIME.2012.10 (cit. on pp. 7, 48, 88, 89).

[GB07] Rodolfo Gómez and Howard Bowman. “Efficient Detection of Zeno Runs in Timed Automata”. In: *FORMATS*. Ed. by Jean-François Raskin and P. S. Thiagarajan. Vol. 4763. Lecture Notes in Computer Science. Salzburg, Austria: Springer, 2007, pp. 195–210. DOI: 10.1007/978-3-540-75454-1_15 (cit. on p. 78).

[GJL10] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. “Learning of event-recording automata”. In: *Theoretical Computer Science* 411.47 (2010), pp. 4029–4054. DOI: 10.1016/j.tcs.2010.07.008 (cit. on p. 65).

[Gla90] Rob J. van Glabbeek. “The Linear Time-Branching Time Spectrum (Extended Abstract)”. In: *CONCUR*. Ed. by Jos C. M. Baeten and Jan Willem Klop. Vol. 458. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, 1990, pp. 278–297. DOI: 10.1007/BFb0039066 (cit. on p. 14).

[Gon+01] Michael González Harbour, J. J. Gutiérrez García, José C. Palencia Gutiérrez, and J. M. Drake Moyano. “MAST: Modeling and Analysis Suite for Real Time Applications”. In: *ECRTS*. Delft, The Netherlands: IEEE Computer Society, 2001, pp. 125–134. DOI: 10.1109/EMRTS.2001.934015 (cit. on p. 90).

[Hen+98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. “What’s Decidable about Hybrid Automata?” In: *Journal of Computer and System Sciences* 57.1 (1998), pp. 94–124. DOI: 10.1006/jcss.1998.1581 (cit. on pp. 28, 46, 98).

[Her+12] Christian Herrera, Bernd Westphal, Sergio Feo Arenis, Marco Muñiz, and Andreas Podelski. “Reducing Quasi-Equal Clocks in Networks of Timed Automata”. In: *FORMATS*. Ed. by Marcin Jurdzinski and Dejan Nickovic. Vol. 7595. Lecture Notes in Computer Science. London, UK: Springer, 2012, pp. 155–170. DOI: 10.1007/978-3-642-33365-1_12 (cit. on p. 68).

[HKW95] Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. “The Expressive Power of Clocks”. In: *ICALP*. Ed. by Zoltán Fülöp and Ferenc Gécseg. Vol. 944. Lecture Notes in Computer Science. Szeged, Hungary: Springer, 1995, pp. 417–428. DOI: 10.1007/3-540-60084-1_93 (cit. on p. 46).

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall, 1985 (cit. on p. 9).

[HPR94] Nicolas Halbwachs, Yann-Éric Proy, and Pascal Raymond. “Verification of Linear Hybrid Systems by Means of Convex Approximations”. In: *SAS*. Ed. by Baudouin Le Charlier. Vol. 864. Lecture Notes in Computer Science. Namur, Belgium: Springer, 1994, pp. 223–237. DOI: 10.1007/3-540-58485-4_43 (cit. on p. 51).

[HSW13] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. “Lazy Abstractions for Timed Automata”. In: *CAV*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Saint Petersburg, Russia: Springer, 2013, pp. 990–1005. DOI: 10.1007/978-3-642-39799-8_71 (cit. on p. 78).

[Hun+02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. “Linear parametric model checking of timed automata”. In: *Journal of Logic and Algebraic Programming* 52-53 (2002), pp. 183–220. DOI: 10.1016/S1567-8326(02)00037-1 (cit. on pp. 7, 8, 16, 17, 19, 25, 26, 34, 35, 37, 44, 55, 79).

[HW16] Christian Herrera and Bernd Westphal. “The Model Checking Problem in Networks with Quasi-Equal Clocks”. In: *TIME*. Ed. by Curtis E. Dyreson, Michael R. Hansen, and Luke Hunsberger. Kongens Lyngby, Denmark: IEEE Computer Society, 2016, pp. 21–30. DOI: 10.1109/TIME.2016.10 (cit. on p. 68).

[JK14] Aleksandra Jovanović and Marta Z. Kwiatkowska. “Parameter Synthesis for Probabilistic Timed Automata Using Stochastic Game Abstractions”. In: *RP*. Ed. by Joël Ouaknine, Igor Potapov, and James Worrell. Vol. 8762. Lecture Notes in Computer Science. Oxford, UK: Springer, 2014, pp. 176–189. DOI: 10.1007/978-3-319-11439-2_14 (cit. on p. 82).

[JL91] Bengt Jonsson and Kim Guldstrand Larsen. “Specification and Refinement of Probabilistic Processes”. In: *LICS*. Amsterdam, The Netherlands: IEEE Computer Society, 1991, pp. 266–277. DOI: 10.1109/LICS.1991.151651 (cit. on p. 82).

[JLR13] Aleksandra Jovanovic, Didier Lime, and Olivier H. Roux. “Synthesis of Bounded Integer Parameters for Parametric Timed Reachability Games”. In: *ATVA*. Ed. by Dang Van Hung and Mizuhito Ogawa. Vol. 8172. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, 2013, pp. 87–101. DOI: 10.1007/978-3-319-02444-8_8 (cit. on p. 99).

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. “Integer Parameter Synthesis for Timed Automata”. In: *IEEE Transactions on Software Engineering* 41.5 (2015), pp. 445–461 (cit. on pp. 7, 12, 18–20, 23, 24, 26, 35, 40, 44, 47, 51, 52, 55, 61, 70, 79, 98, 99).

[Jov13] Aleksandra Jovanović. “Parametric Verification of Timed Systems”. PhD thesis. École Centrale Nantes, 2013 (cit. on p. 23).

[Kho12] Victor Khomenko. *Punf*. http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/. 2012 (cit. on p. 77).

[KMP15] Michał Knapik, Artur Męski, and Wojciech Penczek. “Action Synthesis for Branching Time Logic: Theory and Applications”. In: *ACM Transactions on Embedded Computing* 14.4 (2015). DOI: 10.1145/2746337 (cit. on pp. 81, 82).

[KP12] Michał Knapik and Wojciech Penczek. “Bounded Model Checking for Parametric Timed Automata”. In: *Transactions on Petri Nets and Other Models of Concurrency*. Lecture Notes in Computer Science 6900 (2012). Ed. by Kurt Jensen, Susanna Donatelli, and Jetty Kleijn, pp. 141–159. DOI: 10.1007/978-3-642-29072-5_6 (cit. on pp. 7, 17).

[Kwi+07] Marta Z. Kwiatkowska, Gethin Norman, Jeremy Sproston, and Fuzhi Wang. “Symbolic Model Checking for Probabilistic Timed Automata”. In: *Information and Computation* 205.7 (2007), pp. 1027–1077 (cit. on p. 42).

[Laa+13] Alfons Laarman, Mads Chr. Olesen, Andreas Engelbredt Dalsgaard, Kim Guldstrand Larsen, and Jaco Van De Pol. “Multi-Core Emptiness Checking of Timed Büchi Automata using Inclusion Abstraction”. In: *CAV*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Saint Petersburg, Russia: Springer, 2013, pp. 968–983. DOI: 10.1007/978-3-642-39799-8_69 (cit. on p. 47).

[Lan+17] Rom Langerak, Jaco Van de Pol, Janine N. Post, and Stefano Schivo. "Improving the Timed Automata Approach to Biological Pathway Dynamics". In: *Models, Algorithms, Logics and Tools – Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*. Ed. by Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare. Vol. 10460. Lecture Notes in Computer Science. Springer, 2017, pp. 96–111. DOI: 10.1007/978-3-319-63121-9_5 (cit. on p. 98).

[Li+15] Li Li, Jun Sun, Yang Liu, and Jin Song Dong. "Verifying Parameterized Timed Security Protocols". In: *FM*. Ed. by Nikolaj Bjørner and Frank S. de Boer. Vol. 9109. Lecture Notes in Computer Science. Oslo, Norway: Springer, 2015, pp. 342–359. DOI: 10.1007/978-3-319-19249-9_22 (cit. on p. 99).

[LL73] C. L. Liu and James W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *Journal of the ACM* 20.1 (1973), pp. 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743 (cit. on p. 87).

[LMO06] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. "Model-Checking Timed". In: *FORMATS*. Ed. by Eugene Asarin and Patricia Bouyer. Vol. 4202. Lecture Notes in Computer Science. Paris, France: Springer, 2006, pp. 245–259. DOI: 10.1007/11867340_18 (cit. on p. 10).

[LMT04] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. "Decidability Results for Parametric Probabilistic Transition Systems with an Application to Security". In: *SEFM*. Beijing, China: IEEE Computer Society, 2004, pp. 114–121. DOI: 10.1109/SEFM.2004.12 (cit. on p. 99).

[LMT07] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. "Parametric probabilistic transition systems for system design and analysis". In: *Formal Aspects of Computing* 19.1 (2007), pp. 93–109. DOI: 10.1007/s00165-006-0015-2 (cit. on p. 99).

[LSD16] Li Li, Jun Sun, and Jin Song Dong. "Automated Verification of Timed Security Protocols with Clock Drift". In: *FM*. Ed. by John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou. Vol. 9995. Lecture Notes in Computer Science. Limassol, Cyprus, 2016, pp. 513–530. DOI: 10.1007/978-3-319-48989-6_31 (cit. on p. 99).

[LW82] Joseph Y.-T. Leung and Jennifer Whitehead. "On the complexity of fixed-priority scheduling of periodic, real-time tasks". In: *Performance Evaluation* 2.4 (1982), pp. 237–250. DOI: 10.1016/0166-5316(82)90024-4 (cit. on p. 87).

[Mar11] Nicolas Markey. "Robustness in Real-time Systems". In: *SIES*. Ed. by Iain Bate and Roberto Passerone. Västerås, Sweden: IEEE Computer Society Press, 2011, pp. 28–34. DOI: 10.1109/SIES.2011.5953652 (cit. on p. 51).

[Mer74] Philip Meir Merlin. "A study of the recoverability of computing systems." PhD thesis. University of California, Irvine, CA, USA, 1974 (cit. on p. 72).

[Mil00] Joseph S. Miller. "Decidability and Complexity Results for Timed Automata and Semi-linear Hybrid Automata". In: *HSCC*. Ed. by Nancy A. Lynch and Bruce H. Krogh. Vol. 1790. Lecture Notes in Computer Science. Pittsburgh, PA, USA: Springer, 2000, pp. 296–309. ISBN: 3-540-67259-1. DOI: 10.1007/3-540-46430-1_26 (cit. on pp. 17, 23–25, 29, 32, 44).

[Min67] Marvin L. Minsky. *Computation: finite and infinite machines*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1967. ISBN: 0-13-165563-9 (cit. on pp. 22, 32, 33, 38).

[MWP13] Marco Muñiz, Bernd Westphal, and Andreas Podelski. "Detecting Quasi-equal Clocks in Timed Automata". In: *FORMATS*. Ed. by Víctor A. Braberman and Laurent Fribourg. Vol. 8053. Lecture Notes in Computer Science. Buenos Aires, Argentina: Springer, 2013, pp. 198–212. DOI: 10.1007/978-3-642-40229-6_14 (cit. on p. 68).

[NWY99] Christer Norström, Anders Wall, and Wang Yi. "Timed Automata as Task Models for Event-Driven Systems". In: *RTCSA*. Hong Kong, China: IEEE Computer Society, 1999, pp. 182–189. DOI: 10.1109/RTCSA.1999.811218 (cit. on pp. 9, 91).

[OMG12] OMG. *Unified Modeling Language Superstructure, Version 2.5*. https://www.omg.org/spec/UML/2.5/About-UML/. 2012 (cit. on p. 9).

[OW10] Joël Ouaknine and James Worrell. "Towards a Theory of Time-Bounded Verification". In: *ICALP Part II*. Ed. by Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis. Vol. 6199. Lecture Notes in Computer Science. Springer, 2010, pp. 22–37. DOI: 10.1007/978-3-642-14162-1_3 (cit. on p. 23).

[PG98] José C. Palencia Gutiérrez and Michael González Harbour. "Schedulability Analysis for Tasks with Static and Dynamic Offsets". In: *RTSS*. Madrid, Spain: IEEE Computer Society, 1998, pp. 26–37. DOI: 10.1109/REAL.1998.739728 (cit. on pp. 88, 90).

[Qua+16] Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. "Parameter Synthesis for Markov Models: Faster Than Ever". In: *ATVA*. Ed. by Cyrille Artho, Axel Legay, and Doron Peled. Vol. 9938. Lecture Notes in Computer Science. Chiba, Japan, 2016, pp. 50–67. DOI: 10.1007/978-3-319-46520-3_4 (cit. on p. 99).

[Roc+16] Alexandre Rocca, Thao Dang, Eric Fanchon, and Jean Marc Moulis. "Application of the Reachability Analysis for the Iron Homeostasis Study". In: *HSB*. Ed. by Eugenio Cinquemani and Alexandre Donzé. Vol. 9957. Lecture Notes in Computer Science. Grenoble, France, 2016, pp. 67–84. DOI: 10.1007/978-3-319-47151-8_5 (cit. on p. 98).

[Rod+15] César Rodríguez, Marcelo Sousa, Subodh Sharma, and Daniel Kroening. "Unfolding-based Partial Order Reduction". In: *CONCUR*. Ed. by Luca Aceto and David de Frutos-Escrig. Vol. 42. LIPIcs. Madrid, Spain: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 456–469. DOI: 10.4230/LIPIcs.CONCUR.2015.456 (cit. on pp. 77, 78).

[Rou04] Olivier H.and Didier Lime Roux. "Time Petri Nets with Inhibitor Hyperarcs. Formal Semantics and State Space Computation". In: *ICATPN*. Ed. by Jordi Cortadella and Wolfgang Reisig. Vol. 3099. Lecture Notes in Computer Science. Bologna, Italy: Springer, 2004, pp. 371–390. DOI: 10.1007/978-3-540-27793-4_21 (cit. on p. 74).

[RS13] César Rodríguez and Stefan Schwoon. "Cunf: A Tool for Unfolding and Verifying Petri Nets with Read Arcs". In: *ATVA*. Ed. by Dang Van Hung and Mizuhito Ogawa. Vol. 8172. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, 2013, pp. 492–495. DOI: 10.1007/978-3-319-02444-8_42 (cit. on p. 78).

[San11] Ocan Sankur. "Untimed Language Preservation in Timed Systems". In: *MFCS*. Vol. 6907. Lecture Notes in Computer Science. Warsaw, Poland: Springer, 2011, pp. 556–567. DOI: 10.1007/978-3-642-22993-0_50 (cit. on pp. 39, 43, 75, 85).

[San15] Ocan Sankur. "Symbolic Quantitative Robustness Analysis of Timed Automata". In: *TACAS*. Ed. by Christel Baier and Cesare Tinelli. Vol. 9035. Lecture Notes in Computer Science. London, UK: Springer, 2015, pp. 484–498. DOI: 10.1007/978-3-662-46681-0_48 (cit. on p. 70).

[SBM14] Ocan Sankur, Patricia Bouyer, and Nicolas Markey. "Shrinking timed automata". In: *Information and Computation* 234 (2014), pp. 107–132. DOI: 10.1016/j.ic.2014.01.002 (cit. on pp. 43, 45, 85).

[Sch+12] Stefano Schivo, Jetse Scholma, Brend Wanders, Ricardo A. Urquidi Camacho, Paul E. Van der Vet, Marcel Karperien, Rom Langerak, Jaco Van de Pol, and Janine N. Post. "Modelling biological pathway dynamics with Timed Automata". In: *BIBE*. Larnaca, Cyprus: IEEE Computer Society, 2012, pp. 447–453. DOI: 10.1109/BIBE.2012.6399719 (cit. on p. 98).

[Sch14] Stefan Schwoon. *Mole*. http://lsv.ens-cachan.fr/~schwoon/tools/mole/. 2014 (cit. on p. 77).

[Sch86] Alexander Schrijver. *Theory of linear and integer programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986 (cit. on p. 12).

[SLS98] Danbing Seto, John P. Lehoczky, and Lui Sha. "Task Period Selection and Schedulability in Real-Time Systems". In: *RTSS*. Madrid, Spain: IEEE Computer Society, 1998, pp. 188–198. DOI: 10.1109/REAL.1998.739745 (cit. on p. 90).

[Spu96] Marco Spuri. *Analysis of Deadline Scheduled Real-Time Systems*. Research Report RR-2772. Projet REFLECS. INRIA, 1996 (cit. on p. 87).

[Su+16] Guoxin Su, Yuan Feng, Taolue Chen, and David S. Rosenblum. "Asymptotic Perturbation Bounds for Probabilistic Model Checking with Empirically Determined Probability Parameters". In: *Transactions on Software Engineering* 42.7 (2016), pp. 623–639. DOI: 10.1109/TSE.2015.2508444 (cit. on p. 99).

[Sun+09] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. "PAT: Towards Flexible Verification under Fairness". In: *CAV*. Ed. by Ahmed Bouajjani and Oded Maler. Vol. 5643. Lecture Notes in Computer Science. Grenoble, France: Springer, 2009, pp. 709–714. ISBN: 978-3-642-02657-7. DOI: 10.1007/978-3-642-02658-4_59 (cit. on p. 66).

[TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. "Parametric Model-Checking of Stopwatch Petri Nets". In: *Journal of Universal Computer Science* 15.17 (2009), pp. 3273–3304 (cit. on pp. 8, 49, 72, 73).

[Tra12] Louis-Marie Traonouez. "A Parametric Counterexample Refinement Approach for Robust Timed Specifications". In: *FIT*. Vol. 87. Electronic Proceedings in Theoretical Computer Science. Tallinn, Estonia, 2012, pp. 17–33 (cit. on p. 70).

[Tri99] Stavros Tripakis. "Verifying Progress in Timed Systems". In: *ARTS*. Ed. by Joost-Pieter Katoen. Vol. 1601. Lecture Notes in Computer Science. Bamberg, Germany: Springer, 1999, pp. 299–314 (cit. on p. 78).

[TY01] Stavros Tripakis and Sergio Yovine. "Analysis of Timed Systems Using Time-Abstracting Bisimulations". In: *Formal Methods in System Design* 18.1 (2001), pp. 25–68. ISSN: 0925-9856. DOI: 10.1023/A:1008734703554 (cit. on p. 37).

[TYB05] Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. "Checking Timed Büchi Automata Emptiness Efficiently". In: *Formal Methods in System Design* 26.3 (2005), pp. 267–292. DOI: 10.1007/s10703-005-1632-8 (cit. on p. 78).

[Wan+06] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. "System architecture evaluation using modular performance analysis: a case study". In: *International Journal on Software Tools for Technology Transfer* 8.6 (2006), pp. 649–667. ISSN: 1433-2779. DOI: 10.1007/s10009-006-0019-5 (cit. on p. 90).

[Wan+14] Ting Wang, Jun Sun, Yang Liu, Xinyu Wang, and Shanping Li. "Are Timed Automata Bad for a Specification Language? Language Inclusion Checking for Timed Automata". In: *TACAS*. Ed. by Erika Ábrahám and Klaus Havelund. Vol. 8413. Lecture Notes in Computer Science. Grenoble, France: Springer, 2014, pp. 310–325. DOI: 10.1007/978-3-642-54862-8_21 (cit. on p. 69).

[Wan+15] Ting Wang, Jun Sun, Xinyu Wang, Yang Liu, Yuanjie Si, Jin Song Dong, Xiaohu Yang, and Xiaohong Li. "A Systematic Study on Explicit-State Non-Zenoness Checking for Timed Automata". In: *IEEE Transactions on Software Engineering* 41.1 (2015), pp. 3–18. DOI: 10.1109/TSE.2014.2359893 (cit. on pp. 78, 79).

[ZNL16a] Zhengkui Zhang, Brian Nielsen, and Kim Guldstrand Larsen. "Distributed Algorithms for Time Optimal Reachability Analysis". In: *FORMATS*. Ed. by Martin Fränzle and Nicolas Markey. Vol. 9884. Lecture Notes in Computer Science. Quebec, QC, Canada: Springer, 2016, pp. 157–173. DOI: 10.1007/978-3-319-44878-7_10 (cit. on p. 47).

[ZNL16b] Zhengkui Zhang, Brian Nielsen, and Kim Guldstrand Larsen. "Time optimal reachability analysis using swarm verification". In: *SAC*. Ed. by Sascha Ossowski. Pisa, Italy: ACM, 2016, pp. 1634–1640. DOI: 10.1145/2851613.2851828 (cit. on pp. 47, 68).