Integer Parameter Synthesis for Real-Time Systems

Aleksandra Jovanović, Didier Lime, and Olivier H. Roux

Abstract—We provide a subclass of parametric timed automata (PTA) that we can actually and efficiently analyze, and we argue that it retains most of the practical usefulness of PTA for the modeling of real-time systems. The currently most useful known subclass of PTA, L/U automata, has a strong syntactical restriction for practical purposes, and we show that the associated theoretical results are mixed. We therefore advocate for a different restriction scheme: since in classical timed automata, real-valued clocks are always compared to integers for all practical purposes, we also search for parameter values as bounded integers. We show that the problem of the existence of parameter values such that some TCTL property is satisfied is PSPACE-complete. In such a setting, we can of course synthesize all the values of parameters and we give symbolic algorithms, for reachability and unavoidability properties, to do it efficiently, i.e., without an explicit enumeration. This also has the practical advantage of giving the result as symbolic constraints between the parameters. We finally report on a few experimental results to illustrate the practical usefulness of our approach.

Index Terms—Timed automata, parameters, synthesis, model-checking, real-time systems, symbolic algorithms

1 Introduction

Real-time systems ar e ubiquitous, and to ensure their correct design it seems natural to rely on the mathematical framework provided by formal methods. Within that framework, the model-checking of timed models is becoming ever more efficient. It nevertheless requires a complete knowledge of the system. Consequently, the verification can only be performed after the design stage, when the global system and its environment are known. Getting a complete knowledge of a system is often impossible and even when it is possible, it increases the complexity of the conception and the verification of systems. Moreover, if the model of the system is proved wrong or if the environment changes, this complex verification process must be carried out again. It follows that the use of parametric timed models is certainly a very interesting approach for the design of real-time systems.

However, for general parametric formalisms such as parametric timed automata (PTA), the existence of a parameter value such that some state is reachable is undecidable and there is currently no algorithm that solves the synthesis problem of parameter values except for severely restricted subclasses, whose practical usability is unclear.

It is then a challenging issue to define a subclass of parametric timed automata, which retains enough of its expressive power and such that, for both reachability and unavoidability properties, the existence of parameter values is decidable and for which there exist efficient symbolic synthesis algorithms.

• *The authors are with Ecole Centrale de Nantes - IRCCyN UMR CNRS 6597, Nantes, France. E-mail: aleksandra.jovanovic@cs.ox.ac.uk, {Didier.Lime, olivier-h.roux}@irccyn.ec-nantes.fr.*

Manuscript received 19 June 2013; revised 27 June 2014; accepted 23 Aug. 2014. Date of publication 10 Sept. 2014; date of current version 15 May 2015. Recommended for acceptance by M. Woodside.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2014.2357445

1.1 Related Work

Parametric timed automata have been introduced by Alur et al. in [3], as a way to specify parametric timing constraints. They study the parametric emptiness problem which asks if there exists a parameter valuation such that the automaton has an accepting run. The problem is proven undecidable for PTA that use three clocks and six parameters, and applies to both dense and discrete time domain. In [12], the undecidability proof is extended for parametric timed automata that use only strict inequalities. Further in [18], Hune et al. identify a subclass of PTA, called lower bound/upper bound (L/U) automata, for which the emptiness problem is decidable. However, their model-checking algorithm, that uses Difference Bound Matrix as data structure, might not terminate. Decidability results for L/U automata have been further investigated by Bozzelli and La Torre in [8]. They consider infinite accepting runs and liveness properties, and show that main decision problems such as emptiness, finiteness and universality for the set of parameter valuations are decidable and PSPACE-complete. They also study constrained versions of emptiness and universality, where parameters are constrained by linear systems of equalities and inequalities, and obtain decidability if parameters of different types (lower and upper bound parameters) are not compared in the linear constraint. They show how to compute the explicit representation of the set of parameters, when all the parameters are of the same type (L-automata and U-automata).

An approach for the verification of Parametric timed computation tree logic (PTCTL) formulae has been developed in [36] by Wang, where the problem has been proved decidable. A more general problem is studied in [9], where parameters are allowed both in the model and the desired property (PTCTL formula). The authors show that the model-checking problem is decidable and the parameter synthesis problem is solvable, in discrete time, over a PTA

with one parametric clock, if equality is not allowed in the formulae.

In [4], the authors develop a synthesis algorithm that starts from a reference parameter valuation and derives constraints on parameters, ensuring that the behaviors of PTA are time-abstract equivalent. They prove the termination in the acyclic case (all the traces of the automaton are acyclic), while in the general case, the algorithm is not guaranteed to terminate. Henzinger et al. in [16], study more general, hybrid, systems extended with parameters. Their state-space exploration algorithms have been implemented in the model-checking tool HyTech. In [35], the authors analyze time Petri nets (TPNs) with parameters in timing constraints. A property is given as a PTCTL formula, but their model-checking algorithm consists in analysis of a region graph for each parameter valuation. In [34], the authors extend time Petri nets with inhibitor arcs with parameters, and propose an abstraction of the parametric state-space and semi-algorithms for the parametric synthesis problem, considering simple PTCTL formulae.

1.2 Contributions

L/U-automata can be seen as the most useful subclass of PTA supported by many decidability results for reachability-like properties. We show that the existence of parameter valuations such that a given unavoidability property is satisfied is undecidable though. We also pinpoint some difficulties with the actual synthesis of parameter values for L/U automata and reachability properties.

We therefore propose a different way of subclassing PTA: instead of syntactical restrictions of guards and invariants we propose a novel approach based on restricting the possible values of the parameters. To obtain decidability results, we show that we have to restrict these values to bounded integers. From a practical point of view, the subclass of PTA in such a setting is not that restrictive since the temporal constraints for timed automata (TA) are usually defined on natural (or rational) numbers. It is also uncommon in practice not to have a bound (possibly a quite big one) on the delay modelled by some parameter and if we do not, it might be that the constraint is altogether not needed. Nevertheless, this subclass is restrictive enough to make the problems we address decidable and to allow symbolic synthesis algorithms of parameter values.

We give symbolic algorithms to synthesize the set of all parameters valuations for reachability and unavoidability properties, without having to enumerate all the possibilities. These algorithms are implemented in our tool, Roméo.

Finally, we show that the problem of the existence of bounded integer valuations for PTA such that some property is satisfied is PSPACE-complete for a significant number of properties, which include Timed Computation Tree Logic, and also that lifting either of the boundedness or the integer assumption leads to undecidability even for reachability.

1.3 Organization of the Paper

Section 2 gives the basic definitions related to the formalism of parametric timed automata. Section 3 recalls the main positive results on L/U-automata and gives new negative results that make more precise the practical usefulness of that model. This motivates a different restriction scheme based on limiting the possible values of the parameters. Section 4 presents symbolic algorithms for the synthesis problems when parameter valuations are searched as bounded integers. In its development this section also exhibits semi-algorithms for the general setting and the (unbounded) integer setting. Section 5 gives the computational complexities of the associated problems. Finally, Section 6 discusses the performance in practice of the proposed approach, illustrated on a few small but realistic case-studies. We conclude with Section 7.

2 Parametric Timed Automata

$\mathbb{Z}$ is the set of integers, $\mathbb{N}$ the set of natural numbers and $\mathbb{Q}$ is the set of rational numbers. $\mathbb{R}$ is the set of real numbers. $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers and $\mathbb{R}_{>0} = \mathbb{R}_{\geq 0} \setminus \{0\}$. For any closed interval $[a, b]$ of $\mathbb{R}$ with $a, b \in \mathbb{Z}$, we denote by $[a..b]$ its intersection with $\mathbb{Z}$.

Let X be a finite set. 2^X denotes the powerset of X and $|X|$ the size of X.

A *linear expression* on X is an expression generated by the following grammar, for $k \in \mathbb{Z}$ and $x \in X$: $\lambda ::= k \mid k * x \mid \lambda + \lambda$.

Without loss of generality, we consider *reduced* linear expressions λ in which each element of X occurs at most once and with at most one constant term. We let $\mathsf{Coeff}(\lambda, x)$ denote the coefficient of variable $x \in X$ in λ. If x does not occur in λ then $\mathsf{Coeff}(\lambda, x) = 0$. $\mathsf{Coeff}(\lambda, x)$ is well defined since λ is reduced.

$\wedge$ denotes the logical conjunction. A *linear constraint* on X is an expression generated by the following grammar, with λ a linear expression on X, $\sim \in \{>, \geq\}$: $\gamma ::= \lambda \sim 0 \mid \gamma \wedge \gamma$. We denote by $\mathcal{C}(X)$ the set of linear constraints on X.

Let $V \subseteq \mathbb{R}$. A V-valuation for X is a function from X to V. We denote by V^X the set of V-valuations on X.

For any subset $X' \subseteq X$, and a V-valuation v on X, we define the restriction $v_{|X'}$ of v to X' as the unique V-valuation on X' such that $v_{|X'}(x) = v(x)$ for all $x \in X'$. If Y is a set of valuations on X, then $Y_{|X'}$ denotes its projection on X', i.e., $Y_{|X'} = \{v_{|X'} \mid v \in Y\}$.

For a linear expression (resp. constraint) λ on X and a V-valuation v on $X' \subseteq X$, we denote by $v(\lambda)$ the linear expression (resp. constraint) obtained by replacing in λ each element x of X' by the real value $v(x)$. Note that if $X' = X$ then we obtain a real number (resp. a boolean value).

Given some arbitrary order on X, a valuation can be seen as a real vector of size $|X|$. The set of valuations satisfying some linear constraints is then a *convex polyhedron* of $\mathbb{R}^{|X|}$.

A *zone* is a convex polyhedron defined only by conjunctions of constraints of the form $x - y \sim c$ or $x \sim c$, with $x, y \in X, c \in \mathbb{Z}$ and $\sim \in \{<, \leq, \geq, >\}$.

If Z is a convex polyhedron on variable set X defined by the linear constraints $L_1, \ldots, L_n$, $X' \subseteq X$ and v is a valuation on X', then $v(Z)$ is the convex polyhedron defined by the linear constraints $v(L_1), \ldots, v(L_2)$.

Let X (resp. P) be a finite set. We call *clocks* (resp. *parameters*) the elements of X (resp. P). A *simple* (parametric clock)

constraint γ on X (and P) is a linear constraint on $X \cup P$ such that exactly one element x of X occurs in each conjunct of the expression (not necessarily the same for each conjunct), and $\mathsf{Coeff}(\gamma, x) \in \{-1, 1\}$. We denote by $\mathcal{B}(X, P)$ the set of such simple constraints and $\mathcal{B}'(X, P)$ the set of simple constraints in which the clock variable always has coefficient -1. As before, for any V-valuation v on P, and any simple constraint γ on $X \cup P$, $v(\gamma)$ is the linear constraint on X obtained by replacing each parameter $p \in P$ by the real value $v(p)$.

If v is a parameter valuation (on P) and x a clock valuation (on X), we denote by $\frac{x}{v}$ the valuation on $X \cup P$ such that $\frac{x}{v}_{|P} = v$ and $\frac{x}{v}_{|X} = x$. Similarly, for a set of clock valuations Z, we denote by $\frac{Z}{v}$ the set of valuations x on $X \cup P$ such that $x_{|X} \in Z$ and $x_{|P} = v$.

We further define the null valuation $\vec{0}_X$ on X as $\vec{0}_X(x) = 0, \forall x \in X$. For any subset R of X, and any valuation v on X, we denote by $v[R]$ the valuation on X such that $v[R](x) = 0$ if $x \in R$ and $v[R](x) = v(x)$ otherwise. Finally $v + d$, for $d \geq 0$, is the valuation such that $(v + d)(x) = v(x) + d$ for all $x \in X$.

We now introduce parametric timed automata as an extension of the classical model of timed automata.

Definition 1 (Parametric TA). *A Parametric Timed Automaton $\mathcal{A}$ is a tuple $(L, l_0, \Sigma, X, P, E, \mathsf{Inv})$ where: L is a finite set of* locations*; $l_0 \in L$ is the* initial *location; Σ is a finite set of* actions*; X is a finite set of* clocks*; P is a finite set of* parameters*; $E \subseteq L \times \Sigma \times \mathcal{B}(X, P) \times 2^X \times L$ is a finite set of* edges*: if $(l, a, \gamma, R, l') \in E$ then there is an edge from l to l' with action a, (parametric)* guard *γ and set of clocks to reset R; $\mathsf{Inv} : L \to \mathcal{B}'(X, P)$ assigns a (parametric)* invariant *to each location.*

For any $\mathbb{Q}$-valuation v on P, the structure $v(\mathcal{A})$ obtained from $\mathcal{A}$ by replacing each simple constraint γ by $v(\gamma)$ is a *timed automaton* with invariants [2], [17] (TA). In a similar way, if $e = (l, a, \gamma, R, l')$ is an edge of a PTA $\mathcal{A}$, then $v(e) = (l, a, v(\gamma), R, l'))$ is an edge of a timed automaton $v(\mathcal{A})$.

The behavior of a PTA $\mathcal{A}$ is described by that of all the timed automata obtained by considering all possible valuations of the parameters.

Definition 2 (Semantics of a PTA). *Let $\mathcal{A} = (L, l_0, \Sigma, X, P, E, \mathsf{Inv})$ be a PTA and v be an $\mathbb{R}$-valuation on P. The semantics of $v(\mathcal{A})$ is given by the timed transition system $(Q, q_0, \to)$ with:*

- $Q = \{(l, u) \in L \times \mathbb{R}^X_{\geq 0} \mid u(v(\mathsf{Inv}(l))) \text{ is true}\}$;
- $q_0 = (l_0, \vec{0}_X)$ *($q_0 \in Q$ due to the special form of invariants);*
- *Time transitions.* $(l, u) \xrightarrow{d} (l, u + d)$, *with* $d \geq 0$, *iff* $\forall d' \in [0, d], (l, u + d') \in Q$;
- *Action transitions.* $(l, u) \xrightarrow{a} (l', u')$, *with* $a \in \Sigma$, *iff* $(l, u), (l', u') \in Q$, *there exists an edge* $(l, a, \gamma, R, l') \in E$, $u' = u[R]$ *and* $u(v(\gamma))$ *is true.*

A finite *run* is a finite sequence $\rho = q_1 a_1 q_2 a_2 \dots a_{n-1} q_n$ such that for all i, $q_i \in Q$, $a_i \in \Sigma \cup \mathbb{R}_{\geq 0}$ and $q_i \xrightarrow{a_i} q_{i+1}$. For any run ρ, we define $\mathsf{Edges}(\rho) = e_1 \dots e_m$ as the sequence of edges of the automaton taken in the discrete transitions along the run. We suppose without loss of generality that these edges are indeed thus uniquely defined. A run is *maximal* if it either has an infinite number of discrete actions or cannot be extended by a discrete action. We denote by $\mathsf{Runs}(v(\mathcal{A}))$ the set of runs that start in the initial state of $v(\mathcal{A})$.

We can define several interesting parametric problems on PTA. Among them we can ask: does there exist valuations for the parameters such that some property is satisfied? And, even more interesting, can we compute a finite representation of the set of these valuations? Given a class of problems $\mathcal{P}$ (e.g., reachability, unavoidability, TCTL model-checking, control) these two questions translate into what we respectively call the $\mathcal{P}$-emptiness and the $\mathcal{P}$-synthesis problems:

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

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

In this paper we mainly focus on reachability and unavoidability properties and call the corresponding problems EF and AF. Thus, given a PTA $\mathcal{A}$ and a subset G of its locations, EF-emptiness asks: does there exist a valuation v of the parameters such that G is reachable in $v(\mathcal{A})$ from the initial state? And AF-emptiness asks: does there exist a valuation v of the parameters such that all maximal runs in $v(\mathcal{A})$ from the initial state go through G? The related synthesis problems immediately follow.

In [3], the EF-emptiness problem was proved undecidable for PTA. We give further negative results in the next section.

3 L/U-Automata

We briefly present the proof for undecidability of EF-emptiness for PTA from [3], as we will use it later.

The proof is based on a reduction from a 2-counter machine halting problem, known to be undecidable [29]. Recall that a two counter machine M has a finite number of locations $(l_1, \dots, l_n)$ and two non-negative counters C_1 and C_2, as well as instructions that either decrement, increment or test for zero the value of one counter at a time (each instruction can change the location). A configuration of the machine is a tuple (l, c_1, c_2) where l is a location of the machine, c_1 a value for C_1 and c_2 a value for C_2. The machine halts when it reaches a given location $l_{\texttt{halt}}$, from the initial configuration $(l_1, 0, 0)$, for some values of the counters.

A parametric timed automaton $\mathcal{A}_M$ is constructed in a way that it reaches a corresponding halting location $l_{\texttt{halt}}$ for some parameter valuation iff the 2-counter machine M halts.

$\mathcal{A}_M$ has n locations, each one corresponding to a location of the 2-counter machine, plus some auxiliary locations, uses three clocks x, y, z and two parameters a and b. For every instruction of the 2-counter machine, a path between appropriate locations is added to $\mathcal{A}_M$ (auxiliary locations

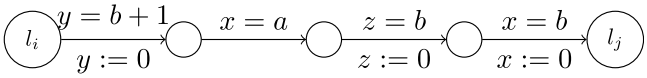

Fig. 1. C_1 increment gadget.

are used to maintain the counter values along the path). Counter values are encoded in the values of clocks $y = b - c_1$ and $z = b - a - c_2$, directly using parameters instead of their valuations ($v(a)$ and $v(b)$) to simplify the writing. The states of $\mathcal{A}_M$ are tuples consisting of a current location and values of clocks $(l, v(x), v(y), v(z))$.

Fig. 1 shows a path to increment counter C_1, [3].

When location l_i is reached, $\mathcal{A}_M$ is in a state $(l_i, x = 0, y = b - c_1, z = b - a - c_2)$ which encodes the configuration (l_i, c_1, c_2) of the machine.

In location l_i we spend exactly $c_1 + 1$ time units before we reach the first auxiliary location. As we reach the location l_j when $x = b$ (the guard between the last auxiliary location and l_j is $x = b$), the duration of the path is b. Since we reset y after $c_1 + 1$ time units, the new value of y is $b - c_1 - 1$.

The new value of clock z when reaching l_j can be calculated similarly, by subtracting the time passed before the reset of z from the total duration of the path $z = b - (a + c_2) = b - a - c_2$, which means that the value of z is preserved.

After we traverse the path, we end up in a state $(l_j, x = 0, y = b - c_1 - 1, z = b - a - c_2)$, which correctly encodes the increment of the first counter and the new configuration of the machine $(l_j, c_1 + 1, c_2)$.

In order to traverse the gadget, the clock values in each location must be such that the guard of the outgoing edge is satisfiable, at least after some time elapsing. For instance, in the second location of the increment gadget, $x = c_1 + 1$, and therefore we must have $c_1 + 1 \leq a$. This generates constraints on the parameters.

If the machine halts, $\mathcal{A}_M$ reaches the corresponding halting location $l_{\texttt{halt}}$, with the set of possible parameter valuations $\{a \geq c_1, b - a \geq c_2\}$, where c_1 and c_2 are the maximal values of the counters. If the machine does not halt, then $\mathcal{A}_M$ does not reach $l_{\texttt{halt}}$ for every parameter valuation. Therefore, the EF-emptiness problem for PTA is undecidable.

The following syntactic subclass of PTA, called L/U-automaton, has been proposed in [18] as a decidable subclass for the EF-emptiness problem. It relies on the notion of upper and lower bounds for parameters:

Definition 3 (Lower and upper bounds). *Let γ be a single conjunct of a simple clock constraint on the set of clocks X and the set of parameters P. Let x be the clock variable occurring in γ. γ is an upper (resp. lower) bound constraint if* $\mathsf{Coeff}(\gamma, x)$ *is negative (resp. positive).*

A parameter p is an upper (resp. lower) bound in γ if $\mathsf{Coeff}(\gamma, p)$ *is positive (resp. negative).*

A parameter p is an upper (resp. lower) bound in the PTA $\mathcal{A}$ if for each conjunct γ of each simple clock constraint in the guards and invariants of $\mathcal{A}$, either $\mathsf{Coeff}(\gamma, p) = 0$ *or p is an upper (resp. lower) bound in γ.*

For example, in a constraint $\gamma = x \geq b \wedge x < a$, b is a lower bound and a is an upper bound in γ.

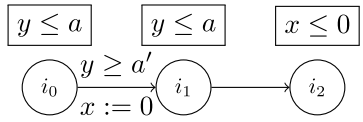

Fig. 2. Parametric Timed Automaton $E(a, a')$ such that, starting from i_0, AFi_2 iff $a = a'$.

Definition 4 (L/U-automaton). *A PTA $\mathcal{A}$ is an L/U-automaton if every parameter is either an upper bound or a lower bound in $\mathcal{A}$.*

A PTA $\mathcal{A}$ is a U-automaton (resp. L-automaton) if every parameter is an upper (resp. lower) bound in $\mathcal{A}$.

3.1 Emptiness

EF-emptiness is PSPACE for L/U-automata [18] and, more generally, emptiness, universality and finiteness of the valuation set are PSPACE-complete for infinite runs acceptance properties [8]. These good results are based on a *monotonicity* property that L/U-automata have: decreasing lower bounds or increasing upper bounds only *add* behaviors. So if we set all lower bounds to 0 and all upper bounds to a large enough constant that we can compute, then the resulting timed automaton contains all the possible behaviors. This makes these automata very well-suited for reachability-like properties. For other properties however this is not enough. For AF properties, increasing lower bounds or decreasing upper bounds can suppress a run that was a counter-example to the property, and then make this property true.

We now indeed prove, with a reduction from the halting problem of 2-counter machines [29], that the AF-emptiness problem for L/U-automata is undecidable.

Theorem 1. *The AF-emptiness problem is undecidable for L/U-automata.*

Proof. As a preliminary, consider PTA $E(a, a')$ in Fig. 2, in which invariants are given in boxes above the corresponding locations. Clearly, starting from i_0, we have $\mathsf{AF}(i_2)$ if and only if $a = a'$, because any run that reaches i_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 i_2. So all runs should enter i_1 with $y = a$, which is the case if and only if $a = a'$.

Using this gadget and adapting those from [3], presented in the beginning of Section 3, we reduce the halting problem of 2-counter machines to the AF-emptiness problem for L/U-automata.

Recall that such a machine has a finite number of locations and two non-negative counters C_1 and C_2, as well as instructions that either decrement, increment or test for zero the value of one counter at a time. Like in the proof of [3], we consider without loss of generality that, in the zero test of some counter, either it succeeds (the counter is indeed 0) and the machine continues or it fails and the machine blocks.

A configuration of the machine is a tuple (l, c_1, c_2) where l is a location of the machine, c_1 a value for C_1 and c_2 a value for C_2. The machine halts when it reaches a given location. The halting problem for 2-counter machines is undecidable [29].

We will have that the machine reaches its $l_{\texttt{halt}}$ location iff for some parameter valuation, a corresponding

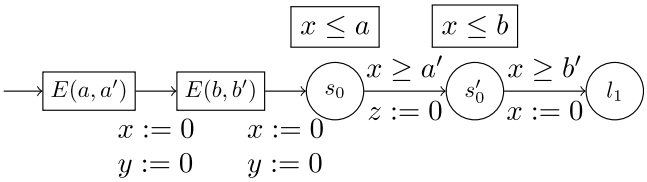

Fig. 3. Setting up the initial configuration.

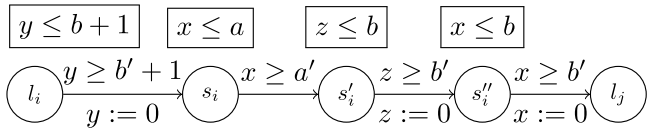

Fig. 4. C_1 increment gadget.

location $l_{\texttt{halt}}$ is unavoidable in the L/U-automaton, i.e., it satisfies the property AF $l_{\texttt{halt}}$.

For each location l_i of the machine we add a corresponding location l_i in the L/U-automaton. The latter also has three clocks x, y, z and four parameters a, b, a' and b'. Parameters a and b are upper bounds and parameters a' and b' are lower bounds.

Throughout the proof, we write the states of the automaton as tuples (l_i, x_i, y_i, z_i) corresponding to location l_i, value x_i for x, y_i for y and z_i for z. Figs. 3 and 4 give the detail of the gadgets we use to encode the operations. They basically correspond to the gadgets of [3] in which the lower bound and upper bound constraints have been separated to obtain an L/U-automaton. Also the upper bounds are expressed as invariants to remove the runs ending with infinite delay in each location.

Each configuration (l_i, c_1, c_2) of the machine is simulated by a state, belonging to some run ρ^*, and of the form $(l_i, 0, y_i, z_i)$, with the following counter encoding: $y_i = b - c_1$ and $z_i = b - a - c_2$. Before we precisely define what run is ρ^* we need to have a closer look at the initialization gadget.

a) Initialization gadget. The initial configuration (location l_1, clock values $x = 0, y_1 = b, z_1 = b - a$) is set using the gadget presented in Fig. 3.

We start, in sequence, by the gadget we have seen before, for both pairs of parameters a, a' and b, b'. All runs go through these first two gadgets iff $a = a'$ and $b = b'$. Furthermore, since AF $l_{\texttt{halt}}$ implies AF s_0, it also implies that $a = a'$ and $b = b'$ in s_0 (and therefore all subsequent locations).

Now, assuming this necessary condition is met, we obtain gadgets quasi-identical to those of [3], except we cannot delay forever in each location. As a consequence, there is only one run generated by this automaton, which is the run ρ^* encoding the counter machine. Let us detail this run in all gadgets.

Initially, we have the state $(s_0, 0, 0, 0)$ of the automaton. By delaying in order to enable the outgoing edge, we obtain state (s_0, a, a, a). In order to take the outgoing edge, the invariant of the target location must be satisfied, which implies $a \le b$. By taking the edge, we obtain state $(s'_0, a, a, 0)$ and, by delay, we get state $(s'_0, b, b, b - a)$. Finally, we take the outgoing edge and get the expected state $(l_1, 0, b, b - a)$.

b) Increment, decrement, zero testing. To simulate increment, we use the gadget given in Fig. 4 for incrementing counter C_1 and going from state l_i to l_j.

We start from some state $(l_i, 0, b - c_1, b - a - c_2)$, for $c_1, c_2 \in \mathbb{N}$ and prove that we reach state $(l_j, 0, b - (c_1 + 1), b - a - c_2)$.

So, first we delay in l_i to enable the outgoing edge and obtain state $(l_i, c_1 + 1, b + 1, b - a - c_2 + c_1 + 1)$. In order to take it, we must have $c_1 + 1 \le a$. Then we take the edge and obtain state $(s_i, c_1 + 1, 0, b - a - c_2 + c_1 + 1)$. Again, we delay and get state $(s_i, a, a - c_1 - 1, b - c_2)$. Then we take the outgoing edge, delay and get $(s'_i, a + c_2, a - c_1 - 1 + c_2, b)$. Again we take the outgoing edge, which implies that $a + c_2 \le b$, and obtain state $(s''_i, a + c_2, a - c_1 - 1 + c_2, 0)$. We delay, which gives state $(s''_i, b, -c_1 - 1 + b, b - a - c_2)$. And finally, by taking the edge to l_j we obtain the expected state $(l_j, 0, b - (c_1 + 1), b - a - c_2)$.

Simulating decrement is similar, using invariant $y \le b - 1$ for l_i instead of $y \le b + 1$ and guard $y \ge b' - 1$ instead of $y \ge b' + 1$. The gadget for incrementing (resp. decrementing) C_2 is obtained in the same way, by replacing invariant $y \le b + 1$ by $y \le b$, guard $y \ge b' + 1$ by $y \ge b'$, invariant $z \le b$ by $z \le b + 1$ (resp. $z \le b - 1$) and guard $z \ge b'$ by $z \ge b' + 1$ (resp. $z' \ge b' - 1$).

Finally, zero-testing C_1 (resp. C_2) can be done with the same gadget, by replacing invariant $y \le b + 1$ by $y \le b$, guard $y \ge b' + 1$ by $y \ge b'$, and by adding conjunct $x \le 0$ (resp. $x \le a$) in the invariant of l_i (resp. s'_i).

If the machine does not halt, then it does not reach location $l_{\texttt{halt}}$ and therefore the automaton does not reach the corresponding location either, so the AF-property never holds for any parameter valuation.

If the machine halts, then no run could be blocked in the zero-testing gadget since zero-testing was always done when the counter was indeed zero and, similarly to [3], the valuations such that the property holds are given by the set $\{a' = a \text{ and } b' = b \text{ and } a \ge c_1^* \text{ and } b - a \ge c_2^*\}$, where c_1^* (resp. c_2^*) is the maximum value of the counter C_1 (resp. C_2) over the finite execution of the machine. These constraints define a convex polyhedron that is non-empty: for instance, $a' = a = c_1^*$ and $b' = b = c_1^* + c_2^*$ belongs to it. □

3.2 Synthesis

In [8] the authors prove that for L-automata and U-automata, the solution to the synthesis problem for infinite runs acceptance properties can be explicitly computed as a linear constraint of size doubly-exponential in the number of parameters. That is to say this solution can be expressed as a finite union of convex polyhedra.

With a different look at the idea used in [8] to prove that the *constrained* (i.e., with initial constraints) emptiness problem for infinite runs acceptance properties is undecidable for L/U-automata, we can express a new and quite strong result on the solution to the EF-synthesis problem for L/U-automata.

Theorem 2. *If it can be computed, the solution to the EF-synthesis problem for L/U-automata cannot be represented using any formalism for which emptiness of the intersection with equality constraints is decidable.*

Proof. We use the same idea as in [8] for proving that *constrained* emptiness for infinite runs acceptance properties is undecidable. Suppose that on the contrary the solution set can be represented using such a formalism. Consider a PTA $\mathcal{A}$. For each parameter p of $\mathcal{A}$ that is used both as an upper bound and a lower bound, replace its occurences as upper bounds by a fresh parameter p_u and its occurrences as lower bounds by a fresh parameter p_l. We therefore obtain an L/U-automaton. Let V be the solution to the EF-synthesis problem for that L/U-automaton. Let V' be the set of equality constraints $p_u = p_l$ for each of the parameters p that were duplicated as p_u and p_l. By hypothesis we can decide if $V \cap V' = \emptyset$ and therefore solve the emptiness problem for $\mathcal{A}$, which contradicts the undecidability of EF-emptiness for PTA. □

Note that, in particular, Theorem 2 rules out the possibilty of computing the solution set as a finite union of polyhedra.

4 Integer Parametric Problems

The decidability results related to emptiness problems for L/U-automata are mixed: properties related to reachability are decidable but very simple properties that are not compatible with the monotonicity property, like unavoidability, are undecidable. As for the actual synthesis of the constraints between parameters that describe the set of valuations that satisfy even the simple case of reachability properties, we have to resort to L- or U-automata, that have severe restrictions regarding their use of parameters.

We therefore advocate for different kinds of restrictions to PTA. Note that with only one irrational constant in the guards of timed automata, reachability is undecidable [28]. For all practical purposes these constants are actually always chosen as integers. Even if we insist on rationals, we can make those integers through adequate scaling and we usually have to since most tools only allow them as integers. So, instead of using syntactical restrictions in the guards and invariants of PTA, we think it makes a lot of sense to search for parameter values as *bounded integers*.

We therefore focus on synthesizing (or just proving the existence of) integer valuations for the parameters: a valuation v on a set X is an integer valuation if $\forall x \in X, v(x) \in \mathbb{Z}$. This induces new emptiness and synthesis problems that we call *integer* problems (e.g., integer EF-emptiness problem).

By insisting that these integer values should be bounded we will be (unsurprisingly) able to make all parametric problems decidable, provided the associated non-parametric problems, obtained by choosing one particular valuation, are decidable of course.

These decidability results are however only interesting for practical purposes if we can solve the corresponding synthesis problems symbolically, i.e., without explicitly enumerating all the possible valuations.

To this end, we first introduce symbolic semi-algorithms to solve the synthesis problems in the general setting (possibly non integer valuations) that are based on a quite straightforward extension of the symbolic zone-based state-space exploration that is ubiquitous for timed automata [24].

4.1 Symbolic States for PTA

We therefore extend the notion of symbolic state of timed automata to PTA, as well as the usual operators associated to them:

Definition 5 (Symbolic state). *A* symbolic state *of a PTA $\mathcal{A}$, with set of clocks X and set of parameters P, is a pair (l, Z) where l is a location of $\mathcal{A}$ and Z is a set of valuations on $X \cup P$.*

For state space computation, we define classical operations on valuation sets:

- future. $Z^{\nearrow} = \{v' \mid v \in Z \wedge v'(x) = v(x) + d, d \geq 0 \text{ if } x \in X; v'(x) = v(x) \text{ if } x \in P\}$;
- past. $Z^{\swarrow} = \{v' \mid v \in Z \wedge v'(x) \geq 0, v'(x) + d = v(x), d \geq 0 \text{ if } x \in X; v'(x) = v(x) \text{ if } x \in P\}$;
- reset of the clock variables in set $R \subseteq X$. $Z[R] = \{v[R] \mid v \in Z\}$;
- initial symbolic state of the PTA $\mathcal{A} = (L, l_0, \Sigma, X, P, E, \mathsf{Inv})$. $\mathsf{Init}(\mathcal{A}) = (l_0, \{v \in \mathbb{R}^{X \cup P} \mid v_{|X} \in \{\vec{0}_X\}^{\nearrow} \wedge v(\mathsf{Inv}(l_0))\})$;
- successor by edge $e = (l, a, \gamma, R, l')$. $\mathsf{Succ}((l, Z), e) = (l', (Z \cap \gamma)[R]^{\nearrow} \cap \mathsf{Inv}(l'))$.

For $S = (l, Z)$, when non-ambiguous, we use S in place of l or Z to simplify the writing a bit. We say that a symbolic state is *reachable* if it can be obtained from $\mathsf{Init}(\mathcal{A})$ by iterative application of the Succ operator for some finite sequence of edges.

$\mathsf{Init}(\mathcal{A})$ is a convex polyhedron and all basic operators preserve convex polyhedra: intersection does so trivially, reset is a projection, future can be done by adding a variable $t \geq 0$, and for all clocks x adding variables x' with constraint $x' = x + t$ and finally eliminating variables t and x for all clocks. These are all basic convex polyhedra-preserving operations. We therefore have the following property.

Property 1. *For any reachable symbolic state (l, Z), Z is a convex polyhedron.*

Additionally, future, reset, intersection with some arbitrary set are all trivially non-decreasing operations with respect to the inclusion of sets of states. So we have:

Property 2. Succ *is non decreasing with respect to the inclusion of sets of states: for all edges e, locations l and sets of states Z and Z', $Z \subseteq Z' \Rightarrow \mathsf{Succ}((l, Z), e) \subseteq \mathsf{Succ}((l, Z'), e)$.*

We also have the following lemma:

Lemma 1. *For any reachable symbolic state (l, Z), for all edges e and valuations v, $v(\mathsf{Succ}((l, Z), e)) = \mathsf{Succ}((l, v(Z)), v(e))$.*

Proof. We prove that the result holds for all suboperations of Succ. First, $v(Z)^{\nearrow} = v(Z^{\nearrow})$. Let $x \in v(Z)^{\nearrow}$. Then there exists $x' \in v(Z)$ and $t \geq 0$ such that $x = x' + t$. $x' \in v(Z)$ so $\frac{x'}{v} \in Z$, and therefore $\frac{x'}{v} + t \in Z^{\nearrow}$. By definition of the future operation, $\frac{x'}{v} + t = \frac{x'+t}{v}$ so $x' + t = x \in v(Z^{\nearrow})$. The other direction and the proof for the reset operation work in the same way.

Now we prove that, $v(Z \cap Z') = v(Z) \cap v(Z')$. Let $x \in v(Z \cap Z')$. Equivalently $\frac{x}{v} = Z \cap Z'$, i.e., $\frac{x}{v} \in Z$ and $\frac{x}{v} \in Z'$, which is exactly $x \in v(Z) \cap v(Z')$. □

We can extend the Succ operator to a sequence of edges $e_1 \dots e_n$ by defining $\mathsf{Succ}((l, Z), e_1 e_2 \dots e_n) = \mathsf{Succ}(\dots \mathsf{Succ}(\mathsf{Succ}((l, Z), e_1), e_2) \dots, e_n)$ and considering that for the empty sequence $\emptyset$, $\mathsf{Succ}(S, \emptyset) = S$.

The following corollaries of Lemma 1 holds:

Corollary 1. *For any reachable symbolic state S, for any edge sequence $e_1, \dots, e_n$:*

$$v(\mathsf{Succ}(S, e_1 \dots e_n) = \mathsf{Succ}(v(S), v(e_1) \dots v(e_n)).$$

Proof. Immediate, by induction on the length of the sequence. □

Corollary 2. *For each parameter valuation v, reachable symbolic state S, and state s, we have $s \in v(S)$ if and only if there is a run of $v(A)$ from the initial state leading to s.*

Proof. S is reachable so there exists an edge sequence $e_1 \dots e_n$ such that $S = \mathsf{Succ}(\mathsf{Init}(\mathcal{A}), e_1 \dots e_n)$. Using Corollary 1, we have $v(S) = \mathsf{Succ}(v(\mathsf{Init}(\mathcal{A})), v(e_1) \dots v(e_n))$. Then, by the classical properties of the zone abstraction on timed automata, $s \in \mathsf{Succ}(v(\mathsf{Init}(\mathcal{A})), v(e_1) \dots v(e_n))$ is equivalent to the existence of a run of $v(\mathcal{A})$ (along edges $e_1 \dots e_n$) from the initial state of $v(\mathcal{A})$ to s, which concludes the proof. □

Finally, for each integer parameter valuation v, $v(\mathcal{A})$ is a timed automaton with integer bounds in clock constraints. So each of its reachable symbolic states is defined by a location and a zone with integer constants, which are topologically closed convex unions of the regions [2] of Alur and Dill (see, e.g., [6]). Therefore such zones have integer vertices. Hence, with Corollary 1, we have the following property.

Property 3. *For any reachable symbolic state (l, Z), if v is an integer parameter valuation then $v(Z)$ is a (convex) zone with integer vertices.*

4.2 Semi-Algorithms for the General Synthesis Problems

The following two algorithms are natural extensions of their timed automata counterpart. The difficulty here is the handling of the parameter valuations.

Let $\mathcal{A}$ be a PTA and G a subset of its locations we want to reach (EF) or make unavoidable (AF).

In both algorithms, conditions are evaluated from top to bottom and M represents a *passed list* of symbolic states. It records the symbolic states that have already been explored on a given path. Initially, M is empty and, the algorithms are called with the initial symbolic state $\mathsf{Init}(\mathcal{A})$ (e.g., for EF, we compute $\mathsf{EF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$).

We compute forward the reachable symbolic states until we reach a location in G or we find a loop on the current path (we compute a symbolic state already present in M). We backpropagate the "good" parameter valuations, i.e., those for which the property is satisfied, through the recursion in the algorithms.

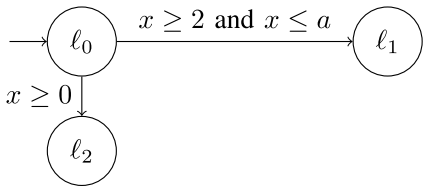

Fig. 5. A PTA in which cutting an edge can be useful for AF.

$\mathsf{Init}(\mathcal{A})$ is a polyhedron and all the operations we perform (successor, projections, etc.) preserve polyhedra so the results of both algorithms are finite unions of polyhedra (but not zones in general).

For EF-synthesis, we basically aggregate the valuations found when reaching the locations in G:

$$\mathsf{EF}_G(S, M) = \begin{cases} S_{|P} & \text{if } S \in G \\ \emptyset & \text{if } S \in M \\ \bigcup\limits_{\substack{e \in E \\ S' = \mathsf{Succ}(S,e)}} \mathsf{EF}_G\big(S', M \cup \{S\}\big) & \text{otherwise.} \end{cases}$$

For AF-synthesis, at a given symbolic state, the "good" valuations, for each outgoing path (hence the intersection in the last line), either allow it to reach G if it can, or cut it off (by being in the complement of its first symbolic state). A path that is cut off is indeed not a path that never reaches G.

Cutting a path, even if it does reach G, may also enable a wider range of parameter values. To illustrate this last point, consider the simple automaton in Fig. 5. The set of goal locations is $G = \{\ell_1, \ell_2\}$, x is a clock and a a parameter. If we never cut paths that lead to G then the result is the intersection of constraints, i.e., $a \geq 2$, while the upper edge could also be cut ($a < 2$) and the AF property still holds, so the correct result is $a \geq 0$.

Furthermore, we need to forbid reaching states from which no transition can be taken, even after some delay. In a symbolic state (l, Z), these correspond to the complement of $\bigcup_{(l,a,g,R,l') \in E} (g \cap Z)^{\nearrow}$. We therefore remove all parameter valuations that allow to reach such a state.

$$\mathsf{AF}_G(S, M) = \begin{cases} S_{|P} & \text{if } S \in G \\ \emptyset & \text{if } S \in M \\ \Big(\bigcap\limits_{\substack{e \in E \\ S' = \mathsf{Succ}(S,e)}} (\mathsf{AF}_G(S', M \cup \{S\}) \cup (\mathbb{R}^P \setminus S'_{|P}))\Big) \setminus \\ \quad (\mathbb{R}^{X \cup P} \setminus (\bigcup_{(l,a,g,R,l') \in E} (g \cap S)^{\nearrow}))_{|P} & \text{otherwise.} \end{cases}$$

Note that it is possible to have a global passed list shared between all paths but this complicates the writing of the algorithms, especially AF.

The following theorem states that EF and AF are semi-algorithms for their respective synthesis problems.

Theorem 3. *For any PTA $\mathcal{A}$ and any subset of its locations G, upon termination, $\mathsf{EF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$ (resp. $\mathsf{AF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$) is the solution to the EF-synthesis (resp. AF-synthesis) problem.*

Proof. In this proof, we use only Lemma 1, its corollary (Corollary 2), and the basic properties of Succ when applied to non-parametric sets of states (i.e., states of timed automata).

Let us consider the possibly infinite directed labeled tree, whose root is labeled by $\mathsf{Init}(\mathcal{A})$ and for every node n, if n is labeled by S, then for all edges e of the PTA, there exists a child n' labeled by $\mathsf{Succ}(S,e)$ iff $\mathsf{Succ}(S,e)$ is not empty. For easier reference, we also label the arc from n to n' by e.

Both algorithms are classical depth-first post-order traversals of that tree.

Now, consider either EF or AF and suppose it has terminated. Then only a finite prefix (a subset closed under the parent relation) T of the infinite tree has been visited and each leaf must correspond to one of the leaf conditions of the algorithms or to the absence of children in the last condition. This means that all leaves n of the tree are labeled by symbolic states S such that:

- either $S = (l, Z)$ and $l \in G$;
- or $S \in M$ and, by construction of M, this means there exists another node on the path from the root to n also labeled by S;
- or S has no successor.

We start by EF and first state the following lemma, the proof of which can be found in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2014.2357445.

Lemma 2. *Let n be a node of T, labeled by some symbolic state S, and such that the subtree rooted at n has depth N. We have: $v \in \mathsf{EF}_G(S, M)$, where M contains the symbolic states labeling nodes on the path from the root, iff there exists a state s in $v(S)$ and a run ρ in $v(\mathcal{A})$, with less than N discrete steps, that starts in s and reaches G.*

Proof. We prove this by induction on N. Note that the tree T is always non-empty (it contains at least the root which is labeled by $\mathsf{Init}(\mathcal{A})$).

- Case of a leaf n labeled by S: the subtree rooted at n has depth 1.
  - if $v \in \mathsf{EF}_G(S, M)$ then the only leaf condition of the algorithm that can be verified is $S = (l, Z)$ and $l \in G$ so for all states in $v(S)$, there is a run with no discrete steps that starts in s and reaches G.
  - if there exists a state $s \in v(S)$ and a run with no discrete steps that starts in s and reaches G, then if l is the location of s, we have $l \in G$, and therefore $v \in \mathsf{EF}_G(S, M)$.
- Case of a non-leaf node n labeled by S: Suppose the subtree rooted at n has depth $k > 1$ and that for all nodes n' with subtree rooted at n' of depth $k' < k$, the property holds.
  - if $v \in \mathsf{EF}_G(S, M)$ then, since n is not a leaf, the third condition of the algorithm must be true: $v \in \bigcup_{\substack{e \in E \\ S' = \mathsf{Succ}(S,e)}} \mathsf{EF}_G(S', M \cup \{S\})$. Equivalently, there exists a successor n' of n, labeled by $S' = \mathsf{Succ}(S, e)$ for some edge e such that $v \in \mathsf{EF}_G(S', M \cup \{S\})$. Since it is a successor of n, n' has depth less than k. So we can use the induction hypothesis: there exists a run with less than $k-1$ discrete steps, starting in some state $s' \in v(S')$ and reaching G in $v(\mathcal{A})$. By Lemma 1, $s' \in \mathsf{Succ}(v(S), v(e))$ so s' has a predecessor s by e in $v(S)$ and we get the expected result.
  - if there exists a run ρ starting in some state $s \in v(S)$ and reaching G, with less than k discrete steps, then this run has at least 1 discrete step otherwise n would be a leaf of T. So we can write it $s \xrightarrow{d} s_d \xrightarrow{a} \rho'$ where a is the action of some edge e. Then ρ' is a run starting from some state $s' \in \mathsf{Succ}(v(S), v(e))$, reaching G and with less than $k-1$ discrete steps. Moreover $s' \in v(S')$ with $S' = \mathsf{Succ}(S, e)$ (by Lemma 1). So we can apply the induction hypothesis and $v \in \mathsf{EF}_G(S', M \cup \{S\})$. Since n is not a leaf, its value for EF_G is given by the last condition, and therefore $v \in \mathsf{EF}_G(S, M)$ by computing the union. □

With Lemma 2, we immediately have that if $v \in \mathsf{EF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$ then there exists a run in $v(\mathcal{A})$ that starts in the initial state and reaches G.

In the other direction, suppose there exists such a run ρ. Then ρ is finite and its last state has a location belonging to G. Let $e_1, \ldots, e_p$ be the edges taken in ρ and consider the branch in the tree T obtained by following this edge sequence on the labels of the arcs in the tree as long as possible. If the whole edge sequence is feasible in T, then the tree T has depth greater or equal to the size of the sequence and we can apply Lemma 2 to obtain that $v \in \mathsf{EF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$. Otherwise, let $S = (l, Z)$ be the symbolic state labeling the last node of the branch, e_k be the first edge in $e_1, \ldots, e_p$ that is not present in the branch and s be the state of ρ just before taking e_k. Using Lemma 1 $v(\mathsf{Succ}(S, e_k))$ is not empty so $\mathsf{Succ}(S, e_k)$ is not empty. Since the node labeled by S has no children in T, it follows that either $l \in G$ or there exists another node on the branch that is labeled by S. In the former case, then we can apply Lemma 2 to the prefix of ρ ending in s and we obtain that $v \in \mathsf{EF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$. In the latter case, by Corollary 2, there exists a run along edges $e_1 \ldots e_m$, with $m < k$, that reaches s in $v(\mathcal{A})$. From that run we can construct another run ρ' by merging with the suffix of ρ that starts from s. ρ' has strictly less discrete actions than ρ and also reaches G and we can repeat the same reasoning as we have just done. We can do this only a finite number of times (because the length of the considered run is strictly decreasing) so at some point we have to be in some of the other cases and we obtain the expected result.

Now we consider the case of AF and we give the following lemma. Again the proof can be found in the appendix, available in the online supplemental material.

Lemma 3. *Let n be a node of T, labeled by some symbolic state S, and such that the subtree rooted at n has depth N. We have: $v \in \mathsf{AF}_G(S, M)$, where M contains the symbolic states labeling nodes on the path from the root, iff for all states s in $v(S)$*

and all maximal runs ρ in $v(\mathcal{A})$ that starts in s, ρ reaches G in less than N discrete steps.

Proof.

- The case of a leaf in T is exactly the same as for EF.
- Case of a non-leaf node n labeled by S: Suppose the subtree rooted at n has depth $k > 1$ and that for all nodes n' with subtree rooted at n' of depth $k' < k$, the property holds.
  - if $v \in \mathsf{AF}_G(S, M)$, since n is not a leaf, the third condition of the algorithm must be true: First, $v \notin (\mathbb{R}^{X \cup P} \setminus (\bigcup_{(l,a,g,R,l') \in E} (g \cap S)))^{\nearrow})_{|P}$. This means that for all states s in $v(S)$, there is at least one edge that can be taken, possibly after some delay. Now for all edges e that can be taken, we further have that, if we note $S' = \mathsf{Succ}(S, e) = (l', Z')$, either $v \in \mathsf{AF}_G(S', M \cup \{S\})$ or $v \in \mathbb{R}^{|P|}_{\geq 0} \setminus Z'_{|P}$. In the latter case, $v(S') = \emptyset$ so, by Lemma 1, the edge e cannot be taken in $v(\mathcal{A})$ from $v(S)$, so only $v \in \mathsf{AF}_G(S', M \cup \{S\})$ holds. As before, the depth of the subtree rooted at the successor node of n labeled by S', is less than $k-1$ and we can therefore apply the induction hypothesis to S' and thus obtain the expected result.
  - if for all states s in $v(S)$ and all maximal runs ρ in $v(\mathcal{A})$ that starts in s, ρ reaches G in less than k discrete steps, then, as for EF, we can write $\rho = s \xrightarrow{d} s_d \xrightarrow{a} \rho'$ where a is the action of some edge e. Then ρ' is a maximal run starting from some state $s' \in \mathsf{Succ}(v(S), v(e)) = v(S')$ with $S' = \mathsf{Succ}(S, e)$ (using Lemma 1), which reaches G in less than $k-1$ discrete steps. For a given e, the set of the first states of the runs ρ', plus those obtained from them by a delay respecting the invariant (who also belong to one of those runs), is exactly $\mathsf{Succ}(v(S), v(e)) = v(S')$ (using once more Lemma 1). So we can apply the induction hypothesis to $v(S')$ and $v \in \mathsf{AF}_G(S', M \cup \{S\})$. Since this is true for all edges e that are first taken by all the runs ρ, $v \in \bigcap_{\substack{e \in E \\ S' = \mathsf{Succ}(S,e)}} \mathsf{AF}_G(S', M \cup \{S\})$. Finally, since all those maximal runs ρ' reach G, there is no state in $v(S')$ that cannot take, possibly after some delay, any discrete transition anymore. Then $v \in \mathsf{AF}_G(S, M)$ using the third condition of the algorithm.

□

Similarly to EF, Lemma 3 immediately implies that if $v \in \mathsf{AF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$ then G is unavoidable in $v(\mathcal{A})$.

In the other direction, suppose that G is unavoidable in $v(\mathcal{A})$. Then all maximal runs starting in the initial state of $v(\mathcal{A})$ reach G. In the same fashion as for EF, we can follow each of those runs along discrete edges in the tree T. If they all reach G "within" the tree T then Lemma 3 gives us the result we expect. If not then there exists a run ρ in $v(\mathcal{A})$ that reaches G by taking discrete edges $e_1 \dots e_p$ but such that only a strict prefix of that sequence is feasible in T. Let e_k be the last feasible edge in the sequence and let $S = (l, Z)$ be the label of the last node in the branch corresponding to the feasible prefix. With the same reasoning as for EF, the only possible cases are that either $l \in G$ or $S \in M$. In the former case, Lemma 3 permits to conlude, so consider the latter case. Then $M \neq \emptyset$ and the feasible prefix contains at least one edge. Let e_m, with $m < k$, be the first edge taken from the previous occurrence of a node labeled by S in the branch corresponding to the feasible prefix. Then $\mathsf{Succ}(S, e_m \dots e_k) = S$ and, using Lemma 1, we can therefore construct an infinite run in $v(\mathcal{A})$ first taking edges $e_1 \dots e_{m-1}$ and then looping on the sequence $e_m \dots e_k$. Since the unavoidability property is satisfied, the prefix of that infinite run up to edge e_k necessarily reaches a location G. And since that prefix is common with ρ, we can use Lemma 3 to conclude the proof. □

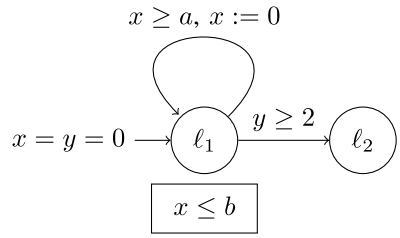

Fig. 6. The PTA $\mathcal{A}_1$ with clocks x and y and parameters a and b.

Example 1. In the PTA $\mathcal{A}_1$ in Fig. 6, after $n > 0$ iterations of the loop, we get the following valuation set $Z_n = \{0 \leq x \leq b, 0 \leq y, a \leq b, 0 \leq na \leq y - x \leq (n+1)b\}$. We can see that we will never have $Z_m = Z_n$ for $m \neq n$ and therefore neither $\mathsf{EF}_{\{\ell_2\}}(\mathsf{Init}(\mathcal{A}_1), \emptyset)$ nor $\mathsf{AF}_{\{\ell_2\}}(\mathsf{Init}(\mathcal{A}_1), \emptyset)$ will terminate.

4.3 Extension for the Integer Synthesis Problems

We now modify the two semi-algorithms to symbolically compute integer valuations. For that we use the notion of integer hull.

Let $n \in \mathbb{N}$ and let Y be a subset of $\mathbb{R}^n$. We denote by $\mathsf{Conv}(Y)$ the *convex hull* of Y, i.e. the intersection of all convex sets containing Y. $\mathsf{IntVects}(Y)$ denotes the subset of all elements of Y with integer coordinates. We call those elements *integer valuations* (or vectors, or points).

Let Z be a convex polyhedron. Z is topologically closed if it can be defined using only non-strict inequalities. The *closure* of Z, denoted by $\overline{Z}$, is the intersection of all closed polyhedra containing Z.

The *integer hull* of a closed polyhedron Z, denoted by $\mathsf{IntHull}(Z)$ is defined as the convex hull of the integer vectors of Z: $\mathsf{IntHull}(Z) = \mathsf{Conv}(\mathsf{IntVects}(Z))$.

An integer vertex of a convex polyhedron is a vertex with integer coordinates. Any (bounded or unbounded) closed convex polyhedron with all its vertices integer is its own integer hull [27].

In the rest of this section, we assume without loss of generality that the polyhedra we consider are topologically

closed. This is not a restriction since any non-closed polyhedron can be represented by a closed polyhedron with one extra dimension [15]. Direct handling of not-necessarily-closed (NNC) polyhedra raises no theoretical issue but would impair the readability of this section. Let us just note that we would need only to define the integer hull of an NNC polyhedron Z as $\mathsf{IntHull}(Z) = Z \cap \mathsf{Conv}(\mathsf{IntVects}(\overline{Z}))$ and, when dealing with the vertices of NNC polyhedra, to also consider the vertices of their closure as described in [5].

We extend IntVects to symbolic states by: $\mathsf{IntVects}((l, Z)) = (l, \mathsf{IntVects}(Z))$ and extend likewise all the other operators on valuation sets.

We now show that to address our integer parametric problems, it is sufficient to consider the integer hulls of the (valuations in the) symbolic states.

We therefore consider the semi-algorithm IEF (resp. IAF) obtained from EF (resp. AF) by replacing all occurrences of the operator Succ by ISucc with $\mathsf{ISucc}((l, Z), e) = \mathsf{IntHull}(\mathsf{Succ}((l, Z), e))$. We also extend ISucc to edge sequences in the same way as for Succ.

Finally, we can state the main result of this section: IEF and IAF are correct semi-algorithms for their respective *integer* synthesis problems.

Theorem 4. *For any PTA $\mathcal{A}$ and any subset of its locations G, upon termination, $\mathsf{IEF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$ (resp. $\mathsf{IAF}_G(\mathsf{Init}(\mathcal{A}), \emptyset)$) is the solution to the integer EF-synthesis (resp. AF-synthesis) problem.*

The proof of Theorem 4 is immediate from the proof of Theorem 3, once we have Lemma 4, which is an equivalent of Lemma 1 for integer valuations.

Lemma 4. *For all* integer *parameter valuations v, symbolic state S reachable through* ISucc *and edge e, $v(\mathsf{ISucc}(S, e)) = \mathsf{Succ}(v(S), v(e))$.*

Proof. Let v be an integer parameter valuation, S be a symbolic state reachable through ISucc and e be an edge in $\mathcal{A}$.

Since S is convex, it certainly holds that $\mathsf{IntHull}(\mathsf{Succ}(S, e)) \subseteq \mathsf{Succ}(S, e)$. And therefore $v(\mathsf{IntHull}(\mathsf{Succ}(S, e))) \subseteq v(\mathsf{Succ}(S, e))$. By Lemma 1, $v(\mathsf{ISucc}(S, e)) \subseteq \mathsf{Succ}(v(S), v(e)))$.

Since v is an integer parameter valuation and (l, Z) is a symbolic reachable state, then by property 3, we have $\mathsf{IntHull}(\mathsf{Succ}(v(S), v(e))) = \mathsf{Succ}(v(S), v(e))$. Let $u \in \mathsf{IntHull}(\mathsf{Succ}(v(S), v(e)))$, then by Lemma 1, $u \in \mathsf{IntHull}(v(\mathsf{Succ}(S, e)))$. Recall that the integer hull of $v(\mathsf{Succ}(S, e))$ is, by definition, the convex hull of the integer vectors of $v(\mathsf{Succ}(S, e))$. So, there exist $z_1, \ldots, z_n \in \mathsf{IntVects}(v(\mathsf{Succ}(S, e)))$ such that u is a convex combination of the z_i's, i.e., there also exist $\lambda_1, \ldots, \lambda_n \in \mathbb{R}_{\geq 0}$ such that $\sum_i \lambda_i = 1$ and $u = \sum_i \lambda_i z_i$. Let x_i be the valuations on $X \cup P$ s.t. $x_{i|P} = v$ and $x_{i|X} = z_i$. Then, since v is an integer valuation, $x_i \in \mathsf{IntVects}(\mathsf{Succ}(S, e))$. Furthermore, if x is the valuation on $X \cup P$ s.t. $x_{|P} = v$ and $x_{|X} = u$, then $x = \sum_i \lambda_i x_i$ and therefore $x \in \mathsf{IntHull}(\mathsf{Succ}(S, e))$ and, by consequence, $u \in v(\mathsf{IntHull}(\mathsf{Succ}(S, e)))$, which concludes the proof. □

Example 2. Let us go back to the PTA $\mathcal{A}_1$ in Fig. 6. After n iterations of the loop, we still get the same valuation set $Z_n = \{0 \leq x \leq b, 0 \leq y, 0 \leq na \leq y - x \leq (n+1)b\}$. This is because Z_n is its own integer hull. So, again neither $\mathsf{IEF}_{\{\ell_2\}}(\mathsf{Init}(\mathcal{A}_1), \emptyset)$ nor $\mathsf{IAF}_{\{\ell_2\}}(\mathsf{Init}(\mathcal{A}_1), \emptyset)$ will terminate.

4.4 Bounded Integer Synthesis Problems

To ensure termination of semi-algorithms IEF and IAF, we now consider that we are searching for bounded integer parameter valuations, i.e., given *a priori* some $M, N \in \mathbb{N}$, we search for integer valuations in $[-M..N]^P$. Again, this induces new emptiness and synthesis problems that we call (M, N)-*bounded integer* problems (e.g., $(100, 100)$-bounded integer EF-emptiness problem).

4.4.1 Bounding the Clocks of PTA

We now show that, without loss of generality, the PTA can be considered to have bounded clocks.

First remark that, in a TA with $|L|$ locations a maximal constant appearing in the constraints of the TA m, and $R(m)$ clock regions (according to the classical definition of [2]), if some location ℓ is reachable, then there exists a run that leads to ℓ and visits at most $|L| \times R(m)$ states. Since it takes at most 1 time unit to go from one region to another, the duration of this run is at most $|L| \times R(m)$ time units. So, if we add invariants $x \leq |L| \times R(m)$ for all clocks x in all the locations of the TA, we obtain an equivalent TA, with respect to location reachability and unavoidability. Since $R(m)$ is non-decreasing with m, this is also true if we increase the value of m.

Now, in our bounded integer parameters setting, we can compute a maximal constant upper (resp. lower) bound on clocks for each parametric upper (resp. lower) bound linear constraint used in the guards and invariants of the automaton: replace the upper bound parameters by their upper (resp. lower) bound and the lower bound parameters by their lower (resp. upper) bound. Let K be the maximum of those maximal constants and of the constants in the non-parametric constraints of the TA. Using the reasoning above, we can then add for all clocks x the invariant $x \leq |L| \times R(K)$ to all locations of our PTA and obtain an equivalent PTA, with respect to location reachability and unavoidability.

4.4.2 Soundness and Correctness of the Algorithms

For a PTA $\mathcal{A}$ with bounded clocks and for any valuation $v \in [-M..N]^P$, $v(\mathcal{A})$ is a TA with bounded clocks for which the finiteness of the number of zones computed with the Succ operator is thus ensured.

Let us define an extension of the Init operator that accepts a bound on the values of the parameters in the initial symbolic state (and therefore in the whole computation): for any $M, N \in \mathbb{N}$, $\mathsf{Init}_{M,N}(\mathcal{A}) = (l_0, \{v \in \mathbb{R}^{X \cup P} \mid v_{|X} \in \{\vec{0}_X\}^{\nearrow} \wedge v(\mathsf{Inv}(l_0)) \text{ and } v_{|P} \in [-M..N]^P\})$.

Theorem 4 can be naturally adapted to this setting in the following form:

Theorem 5. *For any $M, N \in \mathbb{N}$, any PTA $\mathcal{A}$ and any subset of its locations G, upon termination, $\mathsf{IEF}_G(\mathsf{Init}_{M,N}(\mathcal{A}), \emptyset)$ (resp. $\mathsf{IAF}_G(\mathsf{Init}_{M,N}(\mathcal{A}), \emptyset)$) is the solution to the (M, N)-bounded integer EF-synthesis (resp. AF-synthesis) problem.*

The proof of Theorem 5 is immediate from that of Theorem 4 and the following lemma:

Lemma 5. *For any location l, any polyhedron Z on $X \cup P$, and any edge e of $\mathcal{A}$, for any polyhedron C on $X \cup P$ defined only by constraints on P (i.e., for all $v \in C_{|P}$, $v(C) = \mathbb{R}^X$), we have:*

$$\mathsf{Succ}((l, Z \cap C), e) = \mathsf{Succ}((l, Z), e) \cap C.$$

Proof. We first prove that $(Z \cap C)^{\nearrow} = Z^{\nearrow} \cap C$. Let $x \in (Z \cap C)^{\nearrow}$. Then there exists $d \geq 0$ and $x' \in Z \cap C$ such that $x = x' + d$. Since $x' \in Z$, then $x \in Z^{\nearrow}$. Furthermore, since $x'_{|P} = x_{|P}$, $x' \in C$, and $x'_{|P}(C) = \mathbb{R}^X$, we have $x \in C$. The other direction is similar.

We can also prove in the exact same manner that the reset operator behaves similarly because it affects only clocks.

To conclude, recall that if $(l', Z') = \mathsf{Succ}((l, Z), e)$ and $e = (l, a, g, R, l')$ then $Z' = (Z \cap g)[R]^{\nearrow} \cap \mathsf{Inv}(l')$. With the above results and commutativity of intersection, we therefore obtain the expected result. □

4.4.3 Termination of the Algorithms

To prove the termination of our computations, we rely on a few additional lemmas.

First, the following lemma states that the computation of the integer hull of a successor of a symbolic state (l, Z) (reachable from $\mathsf{Init}(\mathcal{A})$), results in the same set as if we would compute the integer hull of a symbolic state (l, Z) at first, and then the integer hull of its successor.

Lemma 6. *For any reachable symbolic state (l, Z) and any edge e:*

$$\mathsf{ISucc}((l, \mathsf{IntHull}(Z)), e) = \mathsf{IntHull}(\mathsf{Succ}((l, Z), e)).$$

Proof. $\mathsf{IntHull}(Z) \subseteq Z$ because Z is convex, and since Succ and $\mathsf{IntHull}$ are non-decreasing, we immediately have $\mathsf{ISucc}((l, \mathsf{IntHull}(Z)), e) \subseteq \mathsf{IntHull}(\mathsf{Succ}((l, Z), e))$.

Let us now consider $\frac{x}{v} \in \mathsf{IntVects}(\mathsf{Succ}((l, Z), e))$. Then, v being an integer parameter valuation, using Lemma 4, we have $x \in \mathsf{Succ}((l, v(Z)), v(e))$. Also, by Property 3, $v(Z)$ has integer vertices so $\mathsf{IntHull}(v(Z)) = v(Z)$. Moreover, $\frac{v(Z)}{v} \subseteq Z$. So, $\mathsf{IntHull}$ being non-decreasing, $\mathsf{IntHull}(\frac{v(Z)}{v}) \subseteq \mathsf{IntHull}(Z)$ and thus $\frac{v(Z)}{v} \subseteq \mathsf{IntHull}(Z)$. Then $v(\frac{v(Z)}{v}) \subseteq v(\mathsf{IntHull}(Z))$, i.e., $v(Z) \subseteq v(\mathsf{IntHull}(Z))$. Succ is also non-decreasing so: $\mathsf{Succ}((l, v(Z)), v(e)) \subseteq \mathsf{Succ}((l, v(\mathsf{IntHull}(Z))), v(e))$. So, using Lemma 1, $x \in v(\mathsf{Succ}((l, \mathsf{IntHull}(Z)), e))$. Then, x being an integer valuation, $\frac{x}{v} \in \mathsf{IntVects}(\mathsf{Succ}((l, \mathsf{IntHull}(Z)), e))$. Finally, Conv being non-decreasing, $\mathsf{IntHull}(\mathsf{Succ}((l, Z), e)) \subseteq \mathsf{IntHull}(\mathsf{Succ}((l, \mathsf{IntHull}(Z)), e))$. □

Then, we extend Lemma 6 to sequences of edges.

Lemma 7. *For any edge sequence $e_1 \dots e_n$:*

$$\mathsf{ISucc}(\mathsf{Init}(\mathcal{A}), e_1 \dots e_n) = \mathsf{IntHull}(\mathsf{Succ}(\mathsf{Init}(\mathcal{A}), e_1 \dots e_n)).$$

Proof. By induction on the length n of the edge sequence: for $n = 0$ the property trivially holds. Suppose the lemma holds for $n \geq 0$ and consider the edge sequence $e_1 \dots e_n e_{n+1}$. $\mathsf{ISucc}(\mathsf{Init}(\mathcal{A}), e_1 \dots e_n e_{n+1}) = \mathsf{ISucc}(\mathsf{ISucc}(\mathsf{Init}(\mathcal{A}), e_1 \dots e_n), e_{n+1})$. By the induction hypothesis this is equal to $\mathsf{ISucc}(\mathsf{IntHull}(\mathsf{Succ}(\mathsf{Init}(\mathcal{A}), e_1 \dots e_n)), e_{n+1})$. Thus, using Lemma 6, this is again equal to $\mathsf{IntHull}(\mathsf{Succ}(\mathsf{Succ}(\mathsf{Init}(\mathcal{A}), e_1 \dots e_n), e_{n+1}))$. And finally, this is indeed equal to $\mathsf{IntHull}(\mathsf{Succ}(\mathsf{Init}(\mathcal{A}), e_1 \dots e_n e_{n+1}))$. □

Finally Lemma 8 states that computing the integer hull of a symbolic state is equivalent to separately computing each of its subsets corresponding to integer parameters and then taking the convex hull of their union.

Lemma 8. *For any reachable symbolic state (l, Z) of the PTA $\mathcal{A}$, $\mathsf{IntHull}(Z) = \mathsf{Conv}(\bigcup_{v \in \mathsf{IntVects}(Z_{|P})} \frac{v(Z)}{v})$.*

Proof. Recall that $\mathsf{IntHull}(Z) = \mathsf{Conv}(\mathsf{IntVects}(Z))$.

Let $x \in \mathsf{IntVects}(Z)$ and $x_{|P}$ be its restriction to parameters. By definition, $x \in \frac{x_{|P}(Z)}{x_{|P}}$ and $x_{|P}$ is an integer valuation. So $x \in \bigcup_{v \in \mathsf{IntVects}(Z_{|P})} \frac{v(Z)}{v}$. Then $\mathsf{IntVects}(Z) \subseteq \bigcup_{v \in \mathsf{IntVects}(Z_{|P})} \frac{v(Z)}{v}$ and, by taking the convex hull, which is non-decreasing with respect to set inclusion, we get $\mathsf{IntHull}(Z) \subseteq \mathsf{Conv}(\bigcup_{v \in \mathsf{IntVects}(Z_{|P})} \frac{v(Z))}{v}$.

Now, let $v \in \mathsf{IntVects}(Z_{|P})$. We certainly have $\frac{v(Z)}{v} \subseteq Z$. So $\mathsf{IntHull}(\frac{v(Z)}{v}) \subseteq \mathsf{IntHull}(Z)$ since $\mathsf{IntHull}$ is non-decreasing. v being an integer parameter valuation, by Property 3, $v(Z)$ and consequently $\frac{v(Z)}{v}$ have integer vertices. It is therefore its own integer hull and $\frac{v(Z)}{v} \subseteq \mathsf{IntHull}(Z)$. So $\bigcup_{v \in \mathsf{IntVects}(Z_{|P})} \frac{v(Z)}{v}) \subseteq \mathsf{IntHull}(Z)$, and since $\mathsf{IntHull}(Z)$ is convex and Conv is non-decreasing, we finally get $\mathsf{Conv}(\bigcup_{v \in \mathsf{IntVects}(Z_{|P})} \frac{v(Z)}{v}) \subseteq \mathsf{IntHull}(Z)$. □

We can finally prove that, in this setting where all the clocks are bounded, the semi-algorithms do terminate:

Theorem 6. *For any $M, N \in \mathbb{N}$, any PTA $\mathcal{A}$ and any subset of its locations G, Algorithms $\mathsf{IEF}_G(\mathsf{Init}_{M,N}(\mathcal{A}), \emptyset)$ and $\mathsf{IAF}_G(\mathsf{Init}_{M,N}(\mathcal{A}), \emptyset)$ terminate.*

Proof. Using Lemma 7 we know that for any edge sequence $e_1 \dots e_n$, $\mathsf{ISucc}(\mathsf{Init}_{M,N}(v(\mathcal{A})), e_1 \dots e_n)$ is actually the integer hull of $(l, Z) = \mathsf{Succ}(\mathsf{Init}_{M,N}(v(\mathcal{A})), e_1 \dots e_n)$.

Then, with Lemma 8, $\mathsf{IntHull}(Z) = \mathsf{Conv}(\bigcup_{v \in \mathsf{IntVects}(Z_{|P})} \frac{v(Z)}{v})$.

For all v in $\mathsf{IntVects}(Z_{|P})$, $v(A)$ is a TA with integer constants and bounded clocks (see Section 4.4.1). It then generates a finite number of different zones. So $\frac{v(Z)}{v}$ takes its values (depending on the edge sequence) in a finite set. Furthermore, since all clocks and parameters are bounded then $\mathsf{IntVects}(Z_{|P})$ is finite and also takes its values in a finite set. Finally, $\mathsf{Conv}(\bigcup_{v \in \mathsf{IntVects}(Z_{|P})} \frac{v(Z)}{v})$ takes its values in a finite set and, consequently, for all possible edge sequences $e_1 \dots e_2$, so does $\mathsf{ISucc}(\mathsf{Init}_{M,N}(v(\mathcal{A})), e_1 \dots e_n)$, which concludes the proof. □

Example 3. Consider once again the PTA $\mathcal{A}_1$ in Fig. 6. We now suppose that both parameters are bounded and take

their values, say in $[0..3]$. Then, as seen in Section 4.4.1, we add the invariants $x \leq 4$ and $y \leq 4$ to both locations (4 is less than the proposed bound but enough in this simple case and keeps the computation understandable). This preserves location-based reachability and unavoidability properties. Now, after $n > 0$ iterations of the loop with the "normal" Succ operator, we have the valuation set $Z_n = \{0 \leq a \leq 3, 0 \leq b \leq 3, a \leq b, 0 \leq x \leq 4, 0 \leq y \leq 4, x \leq b,\ na \leq y - x \leq (n+1)b\}$. If we do not suppose that a and b are integers, we still never have $Z_m = Z_n$ for any $m \neq n$. If we do suppose they are integers, we compute each time $Z'_n = \mathsf{IntHull}(Z_n)$. We have $Z'_0 = Z_0$, $Z'_1 = Z_1 \cap \{y \leq a+3, y \leq b+2\}$, $Z'_2 = Z_2 \cap \{x \leq b-2a+2, y \leq a+3, y-x \leq a+2\}$, $Z'_3 = Z_3 \cap \{a \leq 1, y \leq a+3, y \leq b+3a\}$, $Z'_4 = Z_4 \cap \{y - x = 4a, x \leq 3-3a, x \leq b-a\}$, and when $n \geq 5$, $Z'_n = Z'_{n+1} = \{a = 0, x = y, 0 \leq x \leq b, b \leq 3\}$. And therefore $\mathsf{IEF}_{\{\ell_2\}}(\mathsf{Init}_{0,3}(\mathcal{A}_1), \emptyset)$ terminates and its result is $a \in [0..3]$ and $b \in [1..3]$. Similarly, $\mathsf{IAF}_{\{\ell_2\}}(\mathsf{Init}_{0,3}(\mathcal{A}_1), \emptyset)$ terminates and its result is $a \in [1..3]$ and $b \in [a..3]$.

5 Complexity of the Integer Parametric Problems

When the possible values of the parameters are integer and bounded, we can enumerate all of the possible valuations in exponential time. And therefore, for all classes of problems $\mathcal{P}$ that are in EXPTIME for TA, the $\mathcal{P}$-synthesis problem (and of course the $\mathcal{P}$-emptiness) can be solved in exponential time. Also, since the $\mathcal{P}$ problem for TA is always a special case of the $\mathcal{P}$-emptiness problem for PTA, for problems that are complete for some complexity class containing EXPTIME, we can deduce that the corresponding bounded integer emptiness problem is complete for the same complexity class. For instance, the reachability control problem is EXPTIME-complete for TA [22]. The corresponding parametric emptiness problem is: for a PTA $\mathcal{A}$ with actions partitioned between controllable and uncontrollable, does there exist a parameter valuation v such that there exists a controller for $v(\mathcal{A})$ that enforces the reachability of some location whatever the uncontrollable actions that occur? This problem is therefore EXPTIME-complete for bounded integer parameters.

For simpler problems, we have a better and a bit surprising result, using the classical construction of Savitch giving $\mathrm{PSPACE} = \mathrm{NPSPACE}$ [30]:

Theorem 7. *The $\mathcal{P}$-emptiness problem for PTA with bounded integer parameters is PSPACE-complete for any class of problems $\mathcal{P}$ that is PSPACE-complete for TA.*

Proof. First, by definition, $\mathcal{P}$ for TA is a special case of $\mathcal{P}$-emptiness for PTA with bounded integer valuations (consider a PTA with no parameter). This gives us the PSPACE-hardness.

Now, let $\mathcal{A}$ be a PTA, let ϕ be an instance of $\mathcal{P}$ on $\mathcal{A}$, and let k be the bound on the values of parameters. Consider the non-deterministic Turing machine that:

1) takes $\mathcal{A}$, ϕ and k as input;
2) non-deterministically "guesses" an integer valuation v bounded by k and writes it to the tape;
3) uses the written valuation to overwrite the parameters with their value giving the TA $v(\mathcal{A})$;
4) solves ϕ for that TA;
5) accepts iff the result of the previous step is "yes".

Then the machine obviously accepts iff there exists an integer valuation v bounded by k for which $v(\mathcal{A})$ satisfies ϕ, i.e., it solves the $\mathcal{P}$-emptiness problem for bounded integer parameters.

Now let us look at the complexity. The size of the input is $|\mathcal{A}| + |\phi| + |k|$, using $|.|$ to denote the size in bits of the different objects. There are at most k^p possible valuations, where p is the number of parameters in $\mathcal{A}$. So, storing the valuation at step 2 uses at most $p \times |k|$ additional bits, which is polynomial w.r.t. the size of the input. Step 4 also needs polynomial space by hypothesis. So globally this non-deterministic machine runs in polynomial space. Finally, by Savitch's theorem, we have $\mathrm{PSPACE} = \mathrm{NPSPACE}$ and the expected result. □

In particular the whole TCTL model-checking, including reachability and unavoidability, is PSPACE-complete for TA [1] and as a consequence, the corresponding emptiness problem, which includes EF-emptiness and AF-emptiness, is PSPACE-complete for PTA with bounded integer parameters.

Finally, it is important to remark that we cannot easily lift neither of the boundedness nor the integer assumptions: the EF-emptiness problem for PTA with *bounded rational* parameter values is undecidable [28], and Theorem 8 follows from the undecidability proof of [3]:

Theorem 8. *The EF-emptiness problem for PTA with* possibly unbounded integer *parameter values is undecidable.*

Proof. From the 2-counter machine reduction from [3] (replacing the parameter b_{+1} in the guards by the proper expression $b+1$ and idem for all such parameters):

- When the machine does not halt, the PTA simulating the machine cannot reach the halting location, so there is no rational valuation such that the EF-property holds. And, in particular, there is no integer valuation either;
- When the machine halts, its execution is obviously finite. Let c_1 (resp. c_2) be the biggest value of the counter C_1 (resp. C_2) along that execution. Then the set of valuations such that the EF property holds is $\{a \geq c_1 \text{ and } b - a \geq c_2\}$ which is not empty and contains at least the integer valuation $b = c_1 + c_2$ and $a = c_1$. □

6 On Performance in Practice

6.1 The Tool

We have implemented the synthesis of integer parameter valuations in our tool Roméo [25].

The formal timed model. With its textual input language, Roméo[1] handles a model called clock transition systems (CTS) [19] with timed intervals, which are particularly well-suited for the modeling of real-time systems. This model

1. Notice that the graphical user interface of Roméo remains on Time Petri Nets.

encompasses classical models with implicit clocks such as Time Petri Nets [26] or Duration Kripke structures [23], and models with explicit clocks such as Product Interval Automata [13], [14], which are networks of particular kind of Timed Automata with a single clock which must be reset along every transition. Though best presented in the more well-known framework of parametric timed automata, the method we have proposed in this paper can be naturally adapted to the model of Clock Transition Systems with parametric timed intervals, on which operates ROMÉO.

The symbolic abstraction. On CTS with timed intervals, ROMÉO uses the well known state class abstraction of [7], which is specific to models using timed intervals, and do not require extrapolation. A very interesting feature of the result presented in this paper is that it directly carries over any parametrization of a timed abstraction satisfying Lemma 1, and Properties 1,3 and 2. This is trivially the case for parametric state class abstraction of [34] implemented in ROMÉO for CTSs.

Computing the Integer Hull. In the tool ROMÉO, the computation of the integer hull of a given convex polyhedron is adapted from the computation of Chvátal-Gomory cuts originally formulated in the context of integer linear programming (see, e.g., [31]).

We first shortly introduce Time Petri Nets [26], as we will use them as a model for our second case-study.

Time Petri Nets extend Petri Nets with timing constraints on the firings of transitions. In a TPN, a time interval is associated with each transition. An implicit clock can then be associated with each enabled transition, and gives the elapsed time since it was last enabled. An enabled transition can be fired if its clock value belongs to the interval of the transition. Furthermore, time cannot progress if time elapsing would result in leaving the interval of a transition.

In parametric Time Petri Nets we allow the use of parameters in the time intervals of transitions.

6.2 Case-Studies

The PTA in Fig. 6 demonstrates that it is very easy to find an example for which the symbolic computation does not terminate without the bounded integer parameters restriction but one could object that this PTA models nothing real (if $a = 0$, there are zeno runs for instance).

We now show with two simple but realistic case-studies that this restriction is also useful for real applications. We first describe the two systems:

6.2.1 Task Set Schedulability

We consider a scheduling problem, adapted from [11] for a non-preemptive setting: we have three real-time tasks τ_1, τ_2 and τ_3. τ_1 is periodic with period a and has an execution time $C_1 \in [10, b]$. τ_2 is sporadic: it has only a minimal delay between two activations and that delay is $2a$. The execution time of τ_2 is $C_2 \in [c, d]$, with $c \leq d$. Finally, τ_3 is periodic with period $3a$ and has an execution time $C_3 \in [20, 28]$. These three tasks are scheduled using a non-preemptive[2] priority policy defined by $\tau_1 > \tau_2 > \tau_3$.

2. A running task cannot be interrupted even if another task with a greater priority is ready.

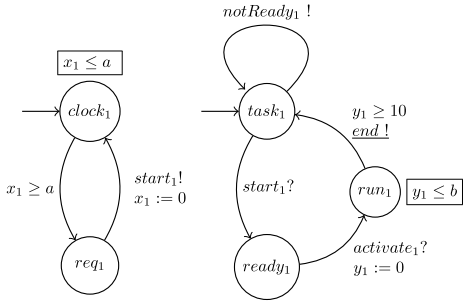

Fig 7.a The periodic request and the model of the task τ_1

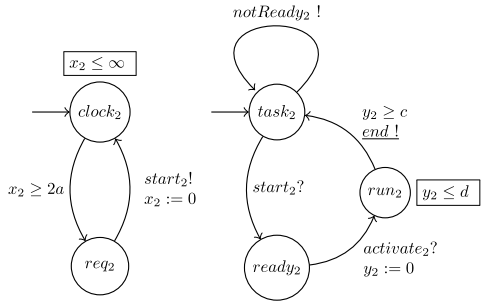

Fig 7.b The sporadic request and the model of the task τ_2

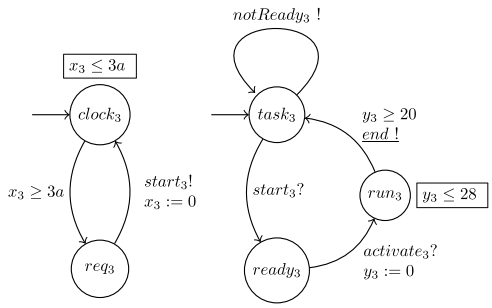

Fig 7.c The periodic request and the model of the task τ_3

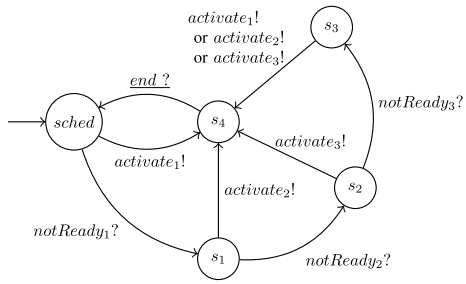

Fig 7.d The scheduler

Fig. 7. A PTA with six clocks and four parameters for non-preemptive scheduling modeling (non-underlined channels are urgent).

We model this problem with a network of parametric timed automata given in Fig. 7. The automata of the network interact with each other by a classical synchronized product *à la Arnold Nivat* where a transition with label $e!$ (respectively $e?$) must be executed simultaneously with one and only one other transition with label $e?$ (respectively $e!$). A synchronization is done as soon as possible (which corresponds to the urgent channel of the tool UPPAAL [6]) except for the synchronization end which is a classical channel.

The models of the tasks and their request are given in Figs. 7a, 7b and 7c. For example, for the task τ_1 in Fig. 7a, as

soon as clock x_1 reaches the value a the synchronization $start_1$ can be performed if the automaton of the task is in state $task_1$. After the synchronization, the automaton of the request is in state $(clock_1, x_1 = 0)$ and the automaton of task τ_1 is in the discrete state $ready_1$. The model of the non-preemptive scheduler is given in Fig. 7d.

We say that the system is schedulable if each task always has at most one instance running, which is a safety property. On our model, this property is verified iff the transitions $start_i$ are always possible in null time meaning for the example of task τ_1, that a state such that the model is in location req_1 with $x_1 > a$ is not reachable.

6.2.2 Alternating Bit Protocol (ABP)

The Alternating Bit Protocol is a network protocol operating at the data link layer that retransmits lost or corrupted messages. This protocol transmits messages between two entities, allowing only one message in transit at a time, over an unreliable transmission medium. Hypotheses on the behavior of the transmission medium are that messages or acknowledgments may be lost in transit. Recovery from losses is done using a time-out and retransmission: each sender records the time at which it sends a message and if an acknowledgment of its delivery does not return within before the time-out, the message is retransmitted. ABP has most of the important features of communication protocols such as TCP and then has been modelled using many extensions of Petri Nets with time [7], [10], [32]. We extend the ABP model of [7] with 6 parameters a, b, c, d, e, f as shown in Fig. 8. The model of [7] corresponds to $a = 5$, $b = 5$, $c = 1$, $d = 1$, $e = 0$ and $f = 2$.

In this net, retransmissions of messages occur at a time given by parameters a and b after the message has been sent. The time for losses of messages and acknowledgments (transitions with no output places) is an interval $[0, 1]$ or $[0, c]$ or $[0, d]$. The time for receptions of messages and acknowledgments is an interval $[0, 1]$. The time for the receiver to send the acknowledgments is $[e, 2]$ or $[0, f]$.

The property that should be verified is that the TPN is one safe. This guarantees that at most one message or acknowledgment is pending at any time and that the transmission medium never holds more than one message or acknowledgment.

6.3 Verification with Roméo

The models we have proposed for both case-studies belong to the class of CTS with parametric timed intervals. In both cases, we have a safety property, which is verified using implementations of the semi-algorithms EF and IEF presented in Section 4. We use a machine with an Intel Core i7 at 2.3 GHz and 8 Gb RAM.

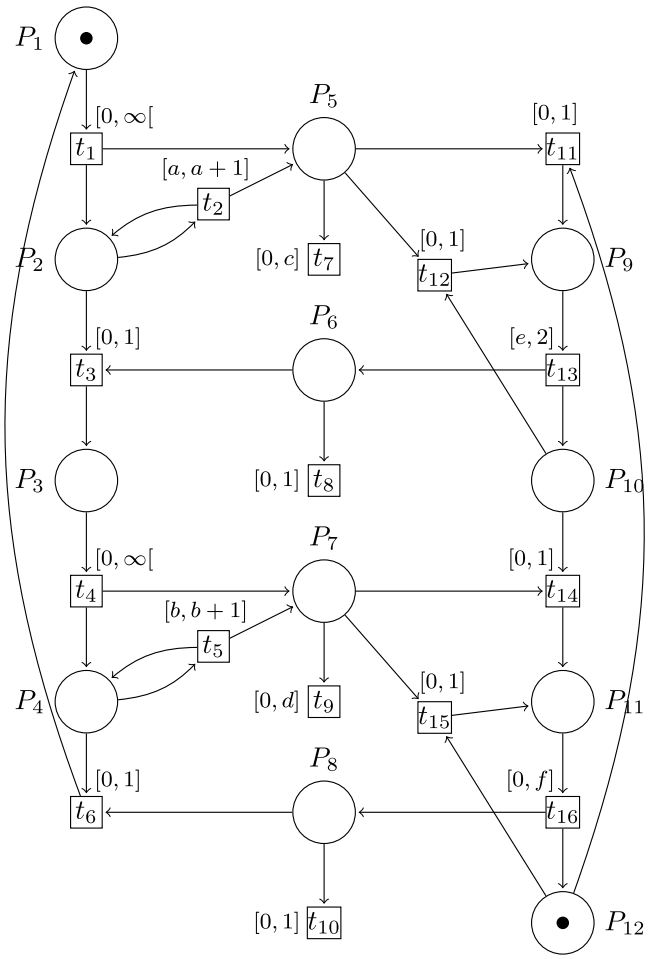

Fig. 8. The parametrized TPN model of the Alternating Bit Protocol.

6.3.1 Obtained Constraints

Using Roméo, and given that all parameters should be non-negative integers, we obtain for the schedulability problem that $a - b \geq 24, b \geq 10, c \leq 28$ when d is equal to 28. As we will see in the next section the computation did not terminate in a reasonable time without fixing the value of d.

For the alternating bit protocol, we obtain the following constraint with all six parameters as non-negative integers:

TABLE 1
Scaling Up the Number of Parameters in the Schedulability Problem

	$a \in [0, \infty)$ $b = 20$ $c = 18$ $d = 28$	$a \in [0, \infty)$ $b \in [10, \infty)$ $c = 18$ $d = 28$	$a \in [0, \infty)$ $b \in [10, \infty)$ $c \in [0, 28]$ $d = 28$	$a \in [0, \infty)$ $b \in [10, \infty)$ $c = 18$ $d \in [18, \infty)$	$a \in [0, \infty)$ $b \in [10, \infty)$ $c \geq 0$ $d \geq c$
IEF Time	1 s	2.8 s	27 s	840 s	DNF
Int. Hull	0.2 s (20%)	0.4 s (14%)	2.9 s (11%)	146 s (17%)	–
IEF Mem.	15 MB	35 MB	153 MB	1,289 MB	–
EF Time	1.5 s	6.4 s	DNF	DNF	DNF
EF Mem.	19.6 MB	55 MB	–	–	–

TABLE 2
Scaling Up the Number of Parameters for the ABP Problem

	a, b	a, b, c	a, b, c, d	a, b, c, d, e	a, b, c, d, e, f
EF Time	0 s	0.2 s	0.7 s	DNF	DNF
EF Mem.	1.6 MB	2.4 MB	4.7 MB	DNF	DNF
IEF Time	0 s	0.2 s	0.6 s	44.7 s	152.7 s
IEF Mem.	1.6 MB	2.5 MB	4.4 MB	75.2 MB	106.9 MB

$$\begin{cases} c = 0 \\ a \geq 4 \\ e \leq 2 \\ b - f \geq 3 \end{cases} \text{ or } \begin{cases} e \leq 2 \\ b - f \geq 3 \\ a \geq 5 \end{cases} \text{ or } \begin{cases} e \leq 2 \\ -b + d + f \leq -2 \\ a \geq 5 \end{cases}$$

$$\text{or } \begin{cases} e = 0 \\ a - c \geq 4 \\ b - f \geq 3 \end{cases} \text{ or } \begin{cases} e = 0 \\ a - c \geq 4 \\ -b + d + f \leq -2 \end{cases} \text{ or } \begin{cases} c = 0 \\ a \geq 4 \\ e \leq 2 \\ -b + d + f \leq -2. \end{cases}$$

6.3.2 Usefulness of the Integer Hull

Tables 1 and 2 provide some insight on the performance of Algorithm IEF and a comparison to Algorithm EF. The only difference in the implementations of the two algorithms is the application or not of the integer hull operator. The table shows the total time for the verifications, the part of it used for computing the integer hull for Algorithm IEF, and the memory consumptions. DNF means that the computation did not finish within 90 min (memory was not a problem here).

Note that in all these cases some parameters are unbounded so termination of Algorithm IEF was actually not guaranteed but it is interesting to remark that the computation did terminate. For both case-studies the use of the integer hull, while sometimes expansive to compute, allows for much better results, in terms of both computation time and memory.

6.3.3 Scaling with Respect to the Bounds on Parameters

With Tables 3 and 4, we illustrate the smooth scaling of our approach with the value of upper bounds (except for very small number of values of parameter). Note that the performance of Algorithm IEF is actually worse when all parameters are bounded (compare with the fourth column of Table 1). This is due to the fact that our implementation uses inclusion for convergence, which is favored by the reduced number of constraints in the absence of upper bounds. In this setting, termination is guaranteed however.

6.3.4 Comparison with an Enumeration of the Parameter Values

To conclude, we comment on the overall usefulness of our approach compared to an explicit enumeration of all the possible parameter values coupled with efficient timed verification.

First, as seen above, our symbolic computation may terminate even if the parameters are not bounded, while explicit enumeration is impossible in this case. This situation occurs in both the case-studies presented here.

Second our approach directly gives a symbolic constraint, which is, in our opinion, more useful than the individual values satisfying the property. Obtaining these constraints from the explicit enumeration technique requires some post-processing.

Finally, in practice, our approach behaves well with respect to the scaling of the bounds on parameters. For an explicit enumeration we can easily see this would not be the case. For the schedulability problem, for instance, the typical verification time with ROMÉO in the timed case (with all parameter values fixed) is reliably 0.1 s. If we consider 100 different possible values for each of the parameters a, b and c, this would give a total computation time of the order of 10,000 s for an explicit enumeration.

We explicitly do this comparison with the state-of-the-art model checker UPPAAL [6] for the schedulabilty problem of Fig. 7. We choose values of parameters leading to both true or false results for the schedulability property. The results are given in Table 5. We can see that there exists a bound from which our symbolic computation behaves faster than an enumerative approach with UPPAAL:

- for one parameter (a), the bound is about 70 values of the parameter;
- for two parameters (a, b), the bound is about 15 values per parameter;
- for three parameters (a, b, c), the bound is under 10 values per parameter.

For the sake of completeness we have also tried to combine explicit enumeration with an efficient decision diagram based model-checker (with a discrete time semantics): we have used the SDD-based *ITS tools*, which is capable of verifying huge state-spaces [33] and has the advantage of providing a parser for ROMÉO models and automatic discrete-time analysis for which it has already given good results. The performance was however not so good here, with the verification of the property on the schedulability problem

TABLE 4
Scaling Up a's Upper Bound for ABP Problem

	$a \leq 10$	$a \leq 100$	$a \leq 1{,}000$	$a \leq 10{,}000$
IEF Time	152.8 s	152.6 s	154 s	153.2 s
Int. Hull	2.3 s (%)	2.3 s (%)	2.3 s (%)	2.3 s (%)
IEF Mem.	108.2 MB	108.2 MB	108.2 MB	108.2 MB

TABLE 3
Scaling Up a's Upper Bound in the Schedulability Problem for $b \in [10, 100], c = 18$ and $d \in [18, 100]$

	$a = 50$	$a \in [40, 50]$	$a \in [0, 100]$	$a \in [0, 1{,}000]$	$a \in [0, 10{,}000]$
IEF Time	17 s	211 s	1,079 s	1,150 s	1,178 s
Int. Hull	2 s (11.7%)	25.7 s (12.1%)	166 s (15.4%)	167 s (14.5%)	168 s (14.3%)
IEF Mem.	121 MB	425 MB	1,598 MB	1,667 MB	1,667 MB

TABLE 5
Comparison with an Explicit Enumeration of Parameter Values Using UPPAAL (with $d = 28$)

	$a \in [30,50]$ $b = 20$ $c = 18$	$a \in [10,100]$ $b = 20$ $c = 18$	$a \in [40,50]$ $b \in [20,30]$ $c = 18$	$a \in [10,50]$ $b \in [10,50]$ $c = 18$	$a \in [40,49]$ $b \in [20,29]$ $c \in [18,22]$	$a \in [10,50]$ $b \in [10,50]$ $c \in [0,28]$	$a \in [10,100]$ $b \in [10,100]$ $c \in [0,28]$
UPPAAL (*enumeration*)	0.34 s	1.4 s	1.8 s	28.1 s	7.9 s	802 s	>2,000 s
ROMEO (*symbolic*)	1.1 s	1.1 s	3.5 s	4.6 s	8.5 s	56.3 s	55.8 s

for a typical instance, namely $a = 50, b = 30, c = d = 18$, taking 74 s for $392{,}962$ states.

Of course, given the good model-checker and a small set for the possible parameter values, an explicit enumeration has good chances to be more efficient. In the general case however, we believe that our approach is more flexible and efficient.

7 Conclusion

We have presented novel results for the parametric verification of timed systems modeled as parametric timed automata. Our new negative results show that even when severely restricting the form of the parametric constraints we encounter undecidabilty for many interesting problems. In particular, we have proved that the AF-emptiness problem is undecidable for L/U-automata (it is still open for U- and L-automata though). So we have proposed instead to restrict the codomain of the parameter valuations to bounded integers.

This is completely orthogonal to previous restriction schemes in the sense that it does not enforce any syntactic restriction on PTA, thus simplifying the modeling activity. Also experimental evidence shows that the symbolic approach we propose to avoid an explicit enumeration of all the possible parameter values is robust to scaling the bounds of the parameters (and improves on convergence even without any bounds in some cases).

Also, in this setting, most problems are of course decidable and we have proved that, for instance, emptiness for TCTL properties, which include reachability and unavoidability, is PSPACE-complete. We have also proved that lifting the boundedness or the integer assumption leads to undecidability. We have exhibited symbolic algorithms that allow to avoid the explicit enumeration of all possible valuations and implemented them in our tool ROMÉO [25].

We have extended the work presented here to timed game automata with parameters, a model that is used for the analysis of control problems on real-time systems, [21]. The challenge was to handle non-convex sets, as the backward exploration, needed for the computation of winning states, creates non-convex zones.

The main problem we are now investigating is whether the symbolic result we obtain, as a finite union of convex polyhedra is dense or not, i.e., wether non-integer points in that result also are valuations satisfying the property. This is undoubtly true for reachability but not so clear for unavoidability or parametric timed reachability games. We may however be able to slightly alter the algorithms to make it true in all cases.

Our other current lines of work on this topic include improving the computation of the integer hulls, the search for less restrictive codomains for parameter valuations, and extension of this work to PTA with stopwatches.

Acknowledgments

The authors thank the anonymous reviewers for their very useful comments. This work was partially funded by the ANR national research programs ImpRo (ANR-2010-BLAN-0317) and PACS (ANR-2014). A preliminary version of this work appeared in [20].

References

[1] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Inf. Comput.*, vol. 104, no. 1, pp. 2–34, May 1993.
[2] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.
[3] R. Alur, T. A. Henzinger, and M. Y. Vardi, "Parametric real-time reasoning," in *Proc. ACM Symp. Theory Comput.*, 1993, pp. 592–601.
[4] E. André, T. Chatain, E. Encrenaz, and L. Fribourg, "An inverse method for parametric timed automata," in *Proc. Workshop Reachability Problems*, Liverpool, United Kingdom, 2008, vol. 223, pp. 29–46.
[5] R. Bagnara, P. Hill, and E. Zaffanella, "Not necessarily closed polyhedra and the double description method," *Formal Aspects Comput.*, vol. 17, pp. 222–257, 2005.
[6] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," in *Proc. 4th Int. School Formal Methods Des. Comput., Commun., Softw. Syst.*, 2004, vol. 3185, pp. 200–236.
[7] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Trans. Softw. Eng.*, vol. 17, no. 3, pp. 259–273, Mar. 1991.
[8] L. Bozzelli and S. L. Torre, "Decision problems for lower/upper bound parametric timed automata," *Formal Methods Syst. Des.*, vol. 35, no. 2, pp. 121–151, 2009.
[9] V. Bruyère and J.-F. Raskin, "Real-time model-checking: Parameters everywhere," *Logical Methods Comput. Sci.*, vol. 3, no. 1, pp. 1–30, 2007.
[10] G. Bucci, L. Carnevali, L. Ridi, and E. Vicario, "Oris: A tool for modeling, verification and evaluation of real-time systems," *Int. J. Softw. Tools Technol. Transfer*, vol. 12, no. 5, pp. 391–403, 2010.
[11] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Timed state space analysis of real-time preemptive systems," *IEEE Trans. Soft. Eng.*, vol. 30, no. 2, pp. 97–111, Feb. 2004.
[12] L. Doyen, "Robust parametric reachability for timed automata," *Inf. Process. Lett.*, vol. 102, no. 5, pp. 208–213, 2007.
[13] D. D'Souza and P. S. Thiagarajan, "Product interval automata: A subclass of timed automata," in *Proc. 19th Conf. Softw. Technol. Theoretical Comput. Sci.*, 1999, pp. 60–71.
[14] D. D'Souza and P. S. Thiagarajan, "Product interval automata," *Sadhana, Indian Academy Sci.*, vol. 27, no. 2, pp. 181–208, 2002.
[15] N. Halbwachs, Y. Proy, and P. Raymond, "Verification of linear hybrid systems by means of convex approximation," in *Proc. 1st Int. Symp. Static Anal.*, 1994, vol. 864, pp. 223–237.
[16] T. A. Henzinger, P.-H. Ho, and H. Wong-toi, "Hytech: A model checker for hybrid systems," *Softw. Tools Technol. Transfer*, vol. 1, pp. 460–463, 1997.
[17] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," *Inform. Comput.*, vol. 111, no. 2, pp. 193–244, 1994.
[18] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager, "Linear parametric model checking of timed automata," *J. Logic Algebraic Program.*, vol. 52/53, pp. 183–220, 2002.

[19] C. Jard, D. Lime, and O. H. Roux, "Clock transition systems," presented at the 21th Int. Workshop Concurrency, Specification Program., Berlin, Germany, Sep. 2012.
[20] A. Jovanović, D. Lime, and O. H. Roux, "Integer parameter synthesis for timed automata," in *Proc. 19th Int. Conf. Tools Algorithms Construction Anal. Syst.*, Mar. 2013, vol. 7795, pp. 401–415.
[21] A. Jovanović, D. Lime, and O. H. Roux, "Synthesis of bounded integer parameters for parametric timed reachability games," in *Proc. 11th Int. Symp. Autom. Technol. Verification Anal.*, Hanoi, Vietnam, Oct. 2013, vol. 8172, pp. 87–101.
[22] M. Jurdzinski and A. Trivedi, "Reachability-time games on timed automata," in *Proc. 34th Int. Colloq. Automata, LangUAGE Program.*, Jul. 2007, vol. 4596, pp. 838–849.
[23] F. Laroussinie, N. Markey, and P. Schnoebelen, "On model checking durational Kripke structures," in *Proc. 5th Int. Conf. Found. Softw. Sci. Comput. Struct.*, 2002, pp. 264–279.
[24] K. G. Larsen, P. Pettersson, and W. Yi, "Model-checking for real-time systems," in *Proc. Fundam. Comput. Theory*, 1995, vol. 965, pp. 62–88.
[25] D. Lime, O. H. Roux, C. Seidner, and L.-M. Traonouez, "Romeo: A parametric model-checker for Petri nets with stopwatches," in *Proc. 15th Int. Conf. Tools Algorithms Construction Anal. Syst.*, Mar. 2009, vol. 5505, pp. 54–57.
[26] P. M. Merlin, "A study of the recoverability of computing systems," Ph.D. dissertation, Dept. Inform. Comput. Sci., Univ. California, Irvine, CA, USA, 1974.
[27] R. Meyer, "On the existence of optimal solutions of integer and mixed-integer programming problems," *Math. Program.*, vol. 7, pp. 223–235, 1974.
[28] J. S. Miller, "Decidability and complexity results for timed automata and semi-linear hybrid automata," in *Proc. 3rd Int. Workshop Hybrid Syst.: Comput. Control*, 2000, vol. 1790, pp. 296–309.
[29] M. L. Minsky, *Computation: Finite and Infinite Machines*. Upper Saddle River, NJ, USA: Prentice-Hall, 1967.
[30] W. J. Savitch. (1970, Apr.). Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.* [Online]. *4(2)*, pp. 177–192 [Online]. Available: http://dx.doi.org/10.1016/S0022-0000(70)80006-X
[31] A. Schrijver, *Theory of Linear and Integer Programming*. New York, NY, USA: Wiley, 1986.
[32] I. Suzuki, "Formal analysis of the alternating bit protocol by temporal petri nets," *IEEE Trans. Softw. Eng.*, vol. 16, no. 11, pp. 1273–1281, Nov. 1990.
[33] Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon, "Hierarchical set decision diagrams and regular models," in *Proc. 15th Int. Conf. Tools Algorithms Construction Anal. Syst.*, Mar. 2009, vol. 5505, pp. 1–15.
[34] L.-M. Traonouez, D. Lime, and O. H. Roux, "Parametric model-checking of stopwatch Petri nets," *J. Univ. Comput. Sci.*, vol. 15, no. 17, pp. 3273–3304, 2009.
[35] I. Virbitskaite and E. Pokozy, "Parametric behaviour analysis for time Petri nets," in *Proc. 5th Int. Conf. Parallel Comput. Technol.*, 1999, vol. 1662, pp. 134–140.
[36] F. Wang, "Parametric timing analysis for real-time systems," *Inf. Comput.*, vol. 130, no. 2, pp. 131–150, Nov. 1996.

Didier Lime is an associate professor in the Computer Science Department at Ecole Centrale de Nantes. He is also a member of the Research Institute on Communications and Cybernetics of Nantes, in the Real-time Systems Group. Since 2012, he has been holding an "habilitation à diriger les recherches" (HDR). His main research interests concern the formal verification and control of timed systems, and their hybrid or parametric extensions.

Olivier H. Roux is full professor at the Ecole Centrale de Nantes (France) and he carries out his research activity at the Research Institute for Communications and Cybernetics of Nantes (IRCCyN - UMR CNRS 6597). He is the head of the Real Time Systems group of IRCCyN. His work deals with the verification and the control of timed systems. He has a particular interest in time Petri nets and timed automata as well as in their stopwatch and parametric extensions.

Aleksandra Jovanović received the PhD degree from Ecole Centrale de Nantes, France, as a member of Research Institute for Communications and Cybernetics of Nantes. She is currently a postdoctoral research assistant at Computer Science Department, University of Oxford. Her research concerns formal verification with focus on quantitative verification of real-time and probabilistic systems and their parametric extensions.