Logical Methods in Computer Science
Volume 16, Issue ?, 2020, pp. ?:1–?:31
https://lmcs.episciences.org/

Submitted Jul. 20, 2018
Published ???. XX, 2020

LANGUAGE PRESERVATION PROBLEMS IN PARAMETRIC TIMED AUTOMATA

ÉTIENNE ANDRÉ, DIDIER LIME, AND NICOLAS MARKEY

Université Paris 13, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France

JFLI, CNRS, Tokyo, Japan

National Institute of Informatics, Tokyo, Japan

Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
URL: `https://lipn.univ-paris13.fr/~andre/`

École Centrale de Nantes, LS2N, CNRS, UMR 6004, Nantes, France
URL: `http://pagesperso.ls2n.fr/~lime-d/`

IRISA, CNRS & Inria & Univ. Rennes, France
URL: `http://people.irisa.fr/Nicolas.Markey/`

Abstract. Parametric timed automata (PTA) are a powerful formalism to model and reason about concurrent systems with some unknown timing delays. In this paper, we address the (untimed) language- and trace-preservation problems: *given a reference parameter valuation, does there exist another parameter valuation with the same untimed language, or with the same set of traces?* We show that these problems are undecidable both for general PTA and for the restricted class of L/U-PTA, even for integer-valued parameters, or over bounded time. On the other hand, we exhibit decidable subclasses: 1-clock PTA, and 1-parameter deterministic L-PTA and U-PTA. We also consider robust versions of these problems, where we additionally require that the language be preserved for all valuations between the reference valuation and the new valuation.

Key words and phrases: timed systems, timing parameters, language preservation, trace preservation, L/U-PTA.

* This work is partially supported by the ANR national research program PACS (ANR-14-CE28-0002), by European projects ERC EQualIS (308087) and FET Cassting (601148), by the ANR-NRF French-Singaporean research program ProMiS (ANR-19-CE25-0015) and by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST. This work is an extension of [AM15].

LOGICAL METHODS IN COMPUTER SCIENCE

DOI:10.23638/LMCS-16(?:?)2020

1. Introduction

Timed Automata. Timed Automata (TA hereafter) were introduced in the 1990's [AD90] as an extension of finite automata with *clock variables*, which can be used to constrain the delays between transitions. Despite this flexibility, TA enjoy efficient algorithms for checking reachability (and many other properties), which makes them a well-suited model for reasoning about real-time systems.

In TA, clock variables are compared to (integer) constants in order to allow or disallow certain transitions. The behaviour of a TA may heavily depend on the *exact* values of the constants, and slight changes in any constant may give rise to very different behaviours. In many cases however, it may be desirable to optimise the values of some of the constants of the automaton, in order to exhibit better performances. The question can then be posed as follows: *given a TA and some of its integer constants, does there exist other values of these constants for which the TA has the exact set of (untimed) behaviours?* We call this problem the *language-preservation problem*.

A special case of this problem occurs naturally in recent approaches for dealing with *robustness* of timed automata [DWDMR08, San11, San13]. The question asked there is whether the behaviour of a timed automaton is preserved when the clock constraints are slightly (parametrically) enlarged. In most of those cases, the existence of a parametric enlargement for which the behaviours are the same as in the original TA has been proved decidable.

For the general problem however, the decidability status remains open. To the best of our knowledge, the only approach to this problem is a procedure (called the *inverse method* [ACEF09]) to compute a dense set of parameter valuations around a reference valuation v_0.

Parametric Timed Automata. In this paper, we address the language-preservation problem using *Parametric Timed Automata* (PTA) [AHV93]. A PTA is a TA in which some of the numerical constants in clock constraints are replaced by symbolic constants (a.k.a. parameters), whose value is not known a priori. The classical problem (sometimes called the *EF-emptiness problem*) in PTA asks whether a given target location of a PTA is reachable for some valuation of the parameter(s). This problem was proven undecidable in various settings: for integer parameter valuations [AHV93, BBLS15], for bounded rational valuations [Mil00], or with only strict constraints (no equality nor closed inequality) [Doy07]. The proofs of these results exist in many different flavours, with various bounds on the number of parameters and clocks needed in the reductions; in contrast, limiting the number of clocks (see *e.g.* [AHV93, BO14, BBLS15]) yields decidability (see [And19] for a survey).

The only non-trivial syntactic subclass of PTA with decidable EF-emptiness problem is the class of L/U-PTA [HRSV02]. These models have the following constraint: each parameter may only be used either always as a lower bound in the clock constraints, or always as an upper bound. For those models, the problems of the emptiness, universality and finiteness (for integer-valued parameters) of the set of parameters under which a target location is reachable, are decidable [HRSV02, BL09]. In contrast, the AF-emptiness problem ("*does there exist a parameter valuation for which a given location is eventually visited along any run?*") is undecidable for L/U-PTA [JLR15]. The EG-emptiness problem ("*does there exist a parameter valuation for which a maximal path remains permanently within a given set of locations?*") exhibits a thin border between decidability and undecidability: the problem is decidable if and only if (rational-valued) parameters are chosen in a closed interval [AL17a].

The full TCTL logic-emptiness ("*does there exist a parameter valuation for which a given TCTL formula holds?*") is undecidable for the simpler class of U-PTA [ALR18], where parameters can only be used as upper bounds in clock constraints.

Our Contributions. In this paper, we first prove that the language-preservation problem (and various related problems) is undecidable in most cases (including for L/U-PTA, or in the time-bounded setting). While it might not look surprising given the numerous undecidability results about PTA, it contrasts with the decidability results proved so far for robustness of TA. In the parametrized approaches to robustness (where the aim is to decide if the language of a timed automaton is preserved under a parametrized perturbation) [DWDMR08, San11, San13], the use of the parameter is much more constrained than what we allow in this paper; this is what makes parametrized robustness analysis decidable.

We then devise a semi-algorithm that solves the language- and trace-preservation problems (and actually synthesizes all parameter valuations yielding the same untimed language (or trace) as a given reference valuation), in the setting of *deterministic* PTA. Finally, we study the decidability of these problems for subclasses of PTA: we prove decidability for PTA with a single clock, and for two subclasses of L/U-PTA with a single parameter.

Outline. Section 2 recalls the necessary preliminaries. Section 3 proves the undecidability of the problems in general. Section 4 introduces a correct semi-algorithm for the trace- and language-preservation synthesis. Section 5 considers the (un)decidability for subclasses of PTA. Section 6 concludes the paper.

2. Definitions

2.1. **Constraints.** We fix a finite set $X = \{x_1, \ldots, x_H\}$ set of real-valued variables (called *clocks* in the sequel). A clock valuation w is a function $w\colon X \to \mathbb{R}_{\geq 0}$. We denote by $\mathbf{0}_X$ the clock valuation assigning 0 to all clocks. We define two operations on clock valuations: for $d \in \mathbb{R}_{\geq 0}$ and a clock valuation w, we let $w+d$ be the valuation w' such that $w'(x) = w(x)+d$ for all $x \in X$. Given a set $R \subseteq X$ and a valuation w, we let $w[R \mapsto 0]$ be the clock valuation w' such that $w'(x) = 0$ if $x \in R$, and $w'(x) = w(x)$ otherwise.

We also fix a finite set $P = \{p_1, \ldots, p_M\}$ of rational-valued variables called *parameters*. A parameter valuation v is a function $v\colon P \to \mathbb{Q}_{\geq 0}$. In the sequel, we will have to handle clocks and parameters together. A valuation is a function $u\colon X \cup P \to \mathbb{R}_{\geq 0}$ such that $u_{|X}$ is a clock valuation and $u_{|P}$ is a parameter valuation.

An *atomic constraint* over X and P is an expression of the form either $x \bowtie p + c$ or $x \bowtie c$ or $p \bowtie c$, where $\bowtie \in \{<, \leq, =, \geq, >\}$, $x \in X$, $p \in P$ and $c \in \mathbb{Z}$. The symbols $\top$ and $\bot$ are also special cases of atomic constraints. Notice that our constraints are a bit more general than in the setting of [AHV93], where only atomic constraints of the form $x \bowtie p$ and $x \bowtie c$ (and $\top$ and $\bot$) were allowed. A *constraint* over X and P is a conjunction of atomic constraints. An *(atomic) diagonal constraint* is a constraint of the form $x - x' \bowtie p + c$ or $x - x' \bowtie c$, where x and x' are two clocks and $\bowtie$, p and c are as in plain atomic constraints. A *generalized constraint* over X and P is a conjunction of atomic constraints and atomic diagonal constraints.

Remark 2.1. *We mainly focus here on continuous time (where clock valuations take real values) and rational-valued parameters, as defined above. However, several of our results remain valid for discrete time (where clock valuations take integer values) and integer-valued parameters. We will mention it explicitly when such is the case.*

A valuation u satisfies an atomic constraint $\varphi\colon x \bowtie p + c$, which we denote $u \models \varphi$, whenever $u(x) \bowtie u(p) + c$. The definition for other constraints is similar. All valuations satisfy $\top$, and none of them satisfies $\bot$. A valuation u satisfies a constraint Φ, denoted $u \models \Phi$ if, and only if, it satisfies all the conjuncts of Φ. A constraint Φ is said to depend on $D \subseteq X \cup P$ whenever for any two valuations u and u' such that $u(d) = u'(d)$ for all $d \in D$, it holds $u \models \Phi$ if, and only if, $u' \models \Phi$. A parameter constraint is a constraint that depends only on P.

Given a partial valuation u and a constraint Φ, we write $u(\Phi)$ for the constraint obtained by replacing each z in the domain $\mathsf{dom}(u)$ of u in Φ with $u(z)$. The resulting constraint depends on $(X \cup P) \setminus \mathsf{dom}(u)$.

We denote by $\Phi{\downarrow_V}$ the *projection* of constraint Φ onto $V \subseteq X \cup P$, *i.e.* the constraint obtained by eliminating the variables not in V. Satisfaction of a projected constraint is defined as: $v \models \Phi{\downarrow_V}$ if, and only if, there exists a valuation u on $X \cup P$ such that $u \models \Phi$ and $u_{|V} = v$. In particular, we will be interested in the projection onto the set P of parameters. Such projections can be computed *e.g.* using Difference Bound Matrices (DBM) [BY03], or Fourier-Motzkin elimination. We also define the *time elapsing* of Φ, denoted by $\Phi^{\uparrow}$, as the *generalized* constraint over X and P obtained from Φ by delaying an arbitrary amount of time: $v \models \Phi^{\uparrow}$ if, and only if, there exists a valuation u on $X \cup P$ and a delay $d \in \mathbb{R}_{\geq 0}$ such that $u \models \Phi$ and $v_{|P} = u_{|P}$ and $v_{|X} = u_{|X} + d$. The time-elapsing of a constraint Φ is a classical computation using polyhedra or parametric extensions of DBM: it can be obtained by preserving all differences between any pair of clocks, preserving lower bounds, relaxing upper bounds on atomic (single-clock) constraints, and preserving all relations between parameters (and constants). Given $R \subseteq X$, we define the *reset* of Φ, denoted by $[\Phi]_R$, as the constraint over X and P obtained from Φ by resetting all clocks in R. Its satisfaction relation is defined as follows: $v \models [\Phi]_R$ if, and only if, there exists a valuation u on $X \cup P$ such that $u \models \Phi$ and $u_{|R} = \mathbf{0}_R$ and $u_{|(P \cup X \setminus R)} = v_{|(P \cup X \setminus R)}$. This is again easily computed using polyhedra, DBM or Fourier-Motzkin elimination.

Example 2.2. *Assume $X = \{x_1, x_2\}$ be a set of clocks and $P = \{p_1, p_2\}$ be a set of parameters. Consider the constraint (involving diagonal constraints) Φ defined by*

$$\Phi \equiv (x_1 = p_1) \wedge (p_1 > p_2) \wedge (x_2 = x_1 - 1) \wedge (x_2 = 3)$$

(the fact that all variables are non-negative is left implicit here). Then, we have

$$\Phi{\downarrow_P} \equiv (p_1 > p_2) \wedge (p_1 = 4).$$

In addition, resetting clock x_2 in Φ gives:

$$[\Phi]_{\{x_2\}} \equiv (x_1 = p_1) \wedge (p_1 > p_2) \wedge (x_2 = 0) \wedge (x_1 = 4).$$

Finally, letting time elapse from valuations satisfying Φ gives:

$$\Phi^{\uparrow} \equiv (x_1 \geq p_1) \wedge (p_1 > p_2) \wedge (x_2 = x_1 - 1) \wedge (x_2 \geq 3) \wedge (p_1 = 4).$$

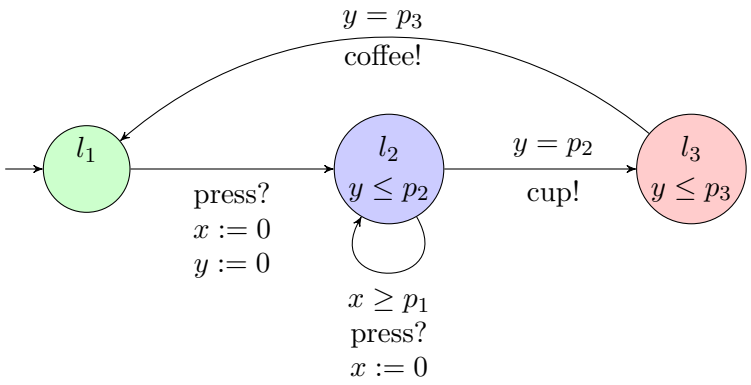


Figure 1: An example of a coffee machine

2.2. **Syntax of Parametric Timed Automata.** Parametric timed automata are an extension of the class of timed automata to the parametric case, where parameters can be used within guards and invariants in place of constants [AHV93].

Definition 2.3. *A* parametric timed automaton *(PTA for short) is a tuple* $\mathcal{A} = \langle \Sigma, L, l_{init}, X, P, I, \rightarrow \rangle$*, where:* Σ *is a finite set of actions;* L *is a finite set of locations;* $l_{init} \in L$ *is the initial location;* X *is a finite set of clocks;* P *is a finite set of parameters;* I *assigns to every* $l \in L$ *a constraint* $I(l)$*, called the* invariant *of* l*;* $\rightarrow$ *is a finite set of edges* (l, g, a, R, l')*, where* $l, l' \in L$ *are the source and destination locations,* g *is a constraint (called* guard *of the transition),* $a \in \Sigma$*, and* $R \subseteq X$ *is a set of clocks to be reset.*

A PTA is *deterministic* if, for all $l \in L$, for all $a \in \Sigma$, there is at most one edge $(l', g, a', R, l'') \in \rightarrow$ with $l' = l$ and $a' = a$. Note that this is a stronger assumption than the usual definition of determinism for TA, which only requires that, for a given action, guards must be mutually exclusive.

A clock is said to be *parametric* if it is compared with a parameter in at least one guard or invariant. Otherwise, it is *non-parametric*.

Example 2.4. *The PTA in Fig. 1 has three locations, three parameters* p_1*,* p_2*,* p_3 *and two clocks* x *and* y*. Both clocks are parametric. This PTA is deterministic.*

2.3. **Semantics of Parametric Timed Automata.** Given a PTA $\mathcal{A} = \langle \Sigma, L, l_{\mathsf{init}}, X, P, I, \rightarrow \rangle$, and a parameter valuation v, $v(\mathcal{A})$ denotes the automaton obtained from $\mathcal{A}$ by substituting every occurrence of a parameter p_i by the constant $v(p_i)$ in the guards and invariants. Then $v(\mathcal{A})$ is a timed automaton [AD90]. The configurations of a timed automaton are the pairs (l, w) where l is a location and w is a clock valuation. Moving from one configuration to another one is allowed depending on the transitions of the PTA, which gives rise to an infinite-state transition system:

Definition 2.5. *Given a PTA* $\mathcal{A} = \langle \Sigma, L, l_{init}, X, P, I, \rightarrow \rangle$*, and a parameter valuation* v*, the semantics of* $v(\mathcal{A})$ *is given by the timed transition system* $\langle Q, q_{init}, \Rightarrow \rangle$ *where* $Q = \{(l, w) \in L \times (\mathbb{R}_{\geq 0})^X \mid w(v(I(l)))$ *evaluates to true*$\}$ *is the set of all valid configurations, with initial*

configuration $q_{init} = (l_{init}, \mathbf{0}_X)$, *and* $((l,w),(d,e),(l',w')) \in \Rightarrow$ *whenever* e *is a transition* $(l,g,a,R,l') \in \rightarrow$ *such that* $w+d \models v(g)$ $w' = (w+d)[R \mapsto 0]$.

A run of a TA is a maximal sequence of consecutive transitions of the timed transition system associated with the TA. For the sake of readability, we usually write runs as $s_0 \overset{d_0,e_0}{\Rightarrow} s_1 \overset{d_1,e_1}{\Rightarrow} \cdots \overset{d_{m-1},e_{m-1}}{\Rightarrow} s_m \cdots$. With *maximal*, we mean that a run may only be finite if its last configuration has no outgoing transition. The timed word associated to a run $s_0 \overset{d_0,e_0}{\Rightarrow} s_1 \overset{d_1,e_1}{\Rightarrow} \cdots \overset{d_{m-1},e_{m-1}}{\Rightarrow} s_m \cdots$ is the (finite or infinite) sequence $(d_i,a_i)_i$ such that for all i, a_i is the action of edge e_i. The corresponding untimed word is the word $(a_i)_i$. The timed (resp. untimed) language of a TA $\mathcal{A}$, denoted by $Lang^t(\mathcal{A})$ (resp. $Lang^u(\mathcal{A})$), is the set of timed (resp. untimed) words associated with runs of this automaton. Similarly, the untimed trace associated with the run $s_0 \overset{d_0,e_0}{\Rightarrow} s_1 \overset{d_1,e_1}{\Rightarrow} \cdots \overset{d_{m-1},e_{m-1}}{\Rightarrow} s_m \cdots$ is the sequence $(l_i,a_i)_i$ s.t. l_i is the location of s_i and a_i is the action of edge e_i. The set of untimed traces of $\mathcal{A}$ is denoted by $Traces(\mathcal{A})$.

A configuration $s = (l,w)$ is said to be reachable in $\mathcal{A}$ under valuation v if s belongs to a run of $v(\mathcal{A})$; a location l is reachable if some configuration of the form (l,w) is reachable.

2.4. **Symbolic Semantics of Parametric Timed Automata.** Following, *e.g.* [HRSV02, ACEF09, JLR15], we now define a symbolic semantics for PTA:

Definition 2.6. *A symbolic state of a PTA* $\mathcal{A}$ *is a pair* (l,C) *where* $l \in L$ *is a location, and* C *is a generalized constraint.*

Given a parameter valuation v, a symbolic state $s = (l,C)$ is v-*compatible* if $v \models C{\downarrow_P}$. The computation of the symbolic state space relies on the Succ operation. The initial symbolic state of $\mathcal{A}$ is $s_{\mathsf{init}} = (l_{\mathsf{init}}, (X=0)^{\uparrow} \wedge I(l_{\mathsf{init}}))$. Given a symbolic state $s = (l,C)$ and a transition $e = (l,g,a,R,l')$, we let

$$\mathsf{Succ}_e(s) = \left\{ (l', C') \;\middle|\; C' = \left([(C \wedge g)]_R\right)^{\uparrow} \cap I(l') \;\wedge\; C' \neq \emptyset \right\}$$

(notice that this is a singleton or the empty set). For transitions e not originating from l, we let $\mathsf{Succ}_e(s) = \emptyset$. We then write $\mathsf{Succ}(s) = \bigcup_{e \in \rightarrow} \mathsf{Succ}_e(s)$. By extension, given a set S of symbolic states, $\mathsf{Succ}(S) = \{s' \mid \exists s \in S \text{ s.t. } s' \in \mathsf{Succ}(s)\}$. Again, this gives rise to an infinite-state transition system, called the *parametric zone graph* later on. A symbolic run of a PTA from some symbolic state s_0 is a maximal alternating sequence $s_0 e_0 s_1 e_1 \cdots$ of symbolic states s_i and edges e_i such that $s_{i+1} = \mathsf{Succ}_{e_i}(s_i)$ for all i. Two runs are said *equivalent* when they correspond to the same sequences of edges (hence the same sequences of locations), but may visit different symbolic states. From now on, a symbolic run of a PTA $\mathcal{A}$ refers to a run starting from the initial symbolic state of $\mathcal{A}$. By extension, a symbolic state of $\mathcal{A}$ is a state belonging to a symbolic run of $\mathcal{A}$.

2.5. **Problems.** In this paper, we address the following two problems:

Definition 2.7. *Given a PTA* $\mathcal{A}$ *and a parameter valuation* v,

- *the* language preservation problem *asks whether there exists another parameter valuation* v' *giving rise to the same untimed language (i.e. such that* $Lang(v(\mathcal{A})) = Lang(v'(\mathcal{A}))$*;*
- *the* trace preservation problem *asks whether there exists another parameter valuation* v' *giving rise to the same set of traces (i.e. such that* $Traces(v(\mathcal{A})) = Traces(v'(\mathcal{A}))$ [ACEF09].

The continuous *versions of those problems additionally require that the language (resp. set of traces) is preserved under any other valuation of the form $\lambda \cdot v + (1-\lambda) \cdot v'$, for $\lambda \in [0,1]$ (with the classical definition of addition and scalar multiplication).*

3. Undecidability of the Preservation Problems in General

3.1. Undecidability of the Language Preservation Problem.

Theorem 3.1. *The language preservation problem for PTA with one parameter is undecidable (both over discrete and continuous time, and for integer and rational parameter valuations).*

Proof. The proof proceeds by a reduction from the halting problem for two-counter machines. We begin with reducing this problem into the classical problem of reachability emptiness ("EF-emptiness") in parametric timed automata, namely: "*is the set of valuations of the parameters for which the target location is reachable empty?*" We then extend the construction to our original problem.

Fix a *deterministic* two-counter machine $\mathcal{M} = \langle S, T \rangle$: such a machine is a finite-state transition system equipped with two counters c_1 and c_2, initially set to zero. The transitions of a two-counter machine can be of two different forms:

- from state s_i, increment c_k and go to s_j. Such a transition is denoted by $(s_i; c_k{+}{+}; s_j)$ in the sequel;
- from state s_i, if $c_k = 0$ then go to s_j, else decrement c_k and go to s_l. Such a transition is denoted $(s_i; s_j; c_k{-}{-}; s_l)$.

In particular, both counters may only take nonnegative values.

The machine starts in state s_0 and halts when it reaches a particular state $s_{\texttt{halt}}$. The halting problem for two-counter machines is undecidable [Min67].

We encode the halting problem of two-counter machines into our problem for PTA. Given a two-counter machine $\mathcal{M}$, we build a PTA whose runs encode the runs of $\mathcal{M}$. Our PTA uses four clocks: clock t will serve as a tick (it will be reset exactly every p time units, where p is the parameter), and we will have a correspondence between a configuration of the timed automaton and a configuration of the two-counter machine every time t is reset; clocks x_1 and x_2 are used to store the values of counters c_1 and c_2 of $\mathcal{M}$, with the correspondence $x_1 = c_1$ and $x_2 = c_2$ when $t = 0$; finally, clock z is used to count the number of steps of the two-counter machine that have been simulated during a computation; this is where our construction differs from the classical ones (*e.g.* [AHV93, JLR15, BBLS15]), as we use the parameter p to bound the length (number of steps) of the computations of $\mathcal{M}$. Notice that p is thus also an upper bound on the values of both c_1 and c_2.

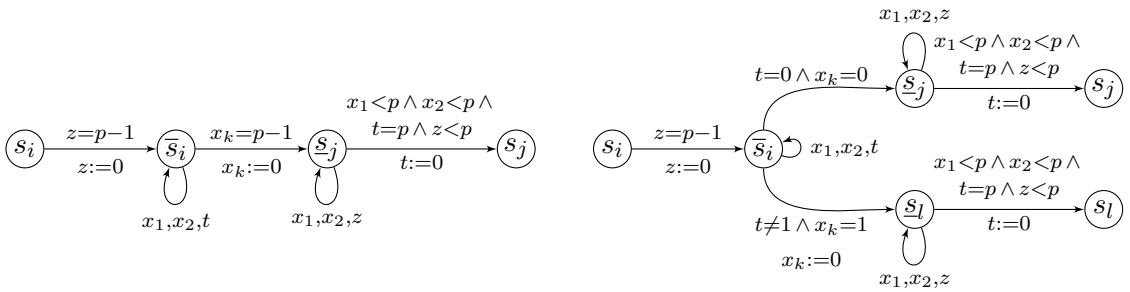
The parametric timed automaton $\mathcal{A}$ associated with $\mathcal{M}$ is defined as follows:

- its set of locations has three copies of the set S of states of $\mathcal{M}$: for each $s \in S$, there is a *main location* with the same name s, and two *intermediary locations* named $\overline{s}$, $\underline{s}$;
- each location of $\mathcal{A}$ has invariants requiring all clocks to never exceed p; all intermediary locations carry two self-loops, resetting clocks x_1 and x_2 when they reach value p. Additionally, locations $\overline{s}$ have a self-loop resetting t when it reaches p, and locations $\underline{s}$ have a self-loop resetting z when it reaches p.

The rough intuition is as follows: the total time elapsed between two consecutive main locations will be p. If in the meantime we reset each clock exactly when it reaches value p (which is the role of the self-loops), then the values encoded by the clocks is unchanged. By resetting some clock one time unit earlier or later, we can encode an increment or decrement of the associated counter.

More precisely, each transition in $\mathcal{M}$ gives rise to several transitions in $\mathcal{A}$:

- first, for each main location s, there is a transition from $\underline{s}$ to s guarded with $t = p$ and resetting t, and a transition from s to $\overline{s}$, guarded with $z = p - 1$ and resetting z; this encodes incrementation of z;
- then, for transitions of $\mathcal{M}$ of the form $(s_i; c_k + +; s_j)$, there is a transition from $\overline{s}_i$ to $\underline{s}_j$ guarded with $x_k = p - 1$ and resetting x_k;
  For transitions of the form $(s_i; s_j; c_k - -; s_l)$, there are two transitions from $\overline{s}_i$: one is guarded with $t = 0 \wedge x_k = 0$, thereby testing that the counter encoded by x_k equals 0; this transitions goes to location $\underline{s}_j$. The second transition[1] is guarded with $t \neq 1 \wedge x_k = 1$; it resets clock x_k and goes to $\underline{s}_l$.



(a) Incrementing c_k (b) Decrementing c_k

Figure 2: Encoding a 2-counter machine. The lists of clocks on self-loops indicate which clocks are reset when they reach value p.

Correctness of this construction is expressed as follows:

Lemma 3.2. *The two-counter machine $\mathcal{M}$ has a halting computation from $(s_0, (c_1 = 0, c_2 = 0))$ if, and only if, there is a run in $v(\mathcal{A})$ reaching the corresponding location $s_{\texttt{halt}}$ from the initial configuration $(s_0, (t = 0, x_1 = 0, x_2 = 0, z = 0))$ in $v(\mathcal{A})$ with $v(p) > 0$.*

Moreover, if $\mathcal{M}$ has no halting computations, then for any v, $v(\mathcal{A})$ eventually reaches a deadlock.

Proof. We prove that the modules for incrementing and decrementing counters correctly implement these operations, as long as z is small enough. Precisely:

- assume that the PTA is in configuration $(s_i, (t = 0, x_1 = x_1^0, x_2 = x_2^0, z = z^0))$ when entering the module encoding transition $(s_i, c_1 + +, s_j)$ incrementing c_1. Assume that $\max(x_1^0, x_2^0) \leq z^0 < p$ (and $t = 0$) when entering that module (which is true initially, and we will prove is preserved when entering the next module). Because clock z cannot be reset in $\overline{s}_i$, at most p time unit can elapse in that location; similarly in $\underline{s}_j$. Hence

[1]The guard $t \neq 1 \wedge x_k = 1$ is not convex, which formally is not allowed in our models; but this is easily encoded by duplicating the transition.

either the automaton reaches location s_j, or it ends up in a deadlock. If the automaton reaches s_j, then the total time elapsed along this run will be either p or $2 \cdot p$, because clock t is initially zero and it equals p at the end of the run; it may have been reset once in the meantime in $\overline{s}_i$, when its value was exactly p. Each resetting self-loop amounts to decrementing the value of its associated clock by p, since it tests if the clock equals p and resets it to zero. Hence the final value of x_2 is unchanged. The transition from s_i to $\overline{s}_i$ amounts to decreasing clock z by $p-1$; all other transitions preserve the value of clock z modulo p, so that in the end the value of z is augmented by 1. The same argument applies to x_1. In the end, if the module is eventually exited, the automaton reaches configuration $(s_j, (t=0, x_1 = x_1^0 + 1, x_2 = x_2^0, z = z^0 + 1))$, as expected.

Conversely, assuming that $\max(x_1^0, x_2^0) \leq z^0 \leq p-1$ (and $t=0$) when entering that module, then there is a path from $(s_i, (t=0, x_1 = x_1^0, x_2 = x_2^0, z = z^0))$ to $(s_j, (t=0, x_1 = x_1^0 + 1, x_2 = x_2^0, z = z^0 + 1))$. If $z^0 > p-1$, then s_i can not be exited, and the automaton ends in a deadlock.

The case of incrementation of c_2 is symmetric.

- similarly, assume that the PTA is in configuration $(s_i, (t=0, x_1 = x_1^0, x_2 = x_2^0, z = z^0))$ when entering the module decrementing c_1. Then counter c_1 equals zero if, and only if, it holds $x_1^0 = 0$ when $t=0$. In that case, the automaton can only proceed to s_j: if that location is ever reached, again the total time elapsed will be an integer multiple of p, and by a similar analysis as above, we get that the automaton will reach configuration $(s_j, (t=0, x_1 = x_1^0 = 0, x_2 = x_2^0, z = z^0 + 1))$.

  On the other hand, if counter c_1 is not zero, *i.e.*, if x_1 is not zero (nor p) when $t=0$, then the automaton can only reach location s_l. With a similar argument as above, we obtain that the automaton will then reach configuration $(s_l, (t=0, x_1 = x_1^0 - 1, x_2 = x_2^0, z = z^0 + 1))$, as required.

  Conversely, if $\max(x_1^0, x_2^0) \leq z^0 \leq p-1$ (and $t=0$) when entering this module, then in both cases ($c_1 = 0$ and $c_1 > 0$) there is a path to the corresponding exit configuration in that module. If on the other hand $z^0 > p-1$, then s_i can not be left. Finally, the case of decrementation of c_2 is symmetric.

From these results, we obtain the fact that if there is a run to location $s_{\texttt{halt}}$ (for some value of p), then it corresponds to a valid halting run of the two-counter machine; conversely, if the two-counter machine has a halting run of length n, then for $v(p) = n+1$, we can build a run in $v(\mathcal{A})$ reaching location $s_{\texttt{halt}}$. Finally, if the two-counter machine has no halting computation, then eventually the value of clock z will exceed $p-1$ when entering a module, which will result in a deadlock. □

We now explain how to adapt this construction to the language preservation problem. The idea is depicted on Fig. 3 (where all transitions are labeled with the same letter a): when $v(p) = 0$, the automaton accepts the untimed language $\{a^\omega\}$. Notice that the guard $p = 0$ in the automaton can be encoded by requiring $t = 0 \wedge t = p$. On the other hand, when $v(p) > 0$, we have to enter the main part of the automaton $\mathcal{A}$, and mimic the two-counter machine. From our construction above, if the run of the two-counter machine is halting run, then for some value $v(p)$, location $s_{\texttt{halt}}$, and then s_∞, will be reached, and the untimed language will be the same as when $v(p) = 0$. Conversely, if the two-counter machine does not halt, then for any value of $v(p)$, the automaton will reach a deadlock, and it will not accept a^ω.

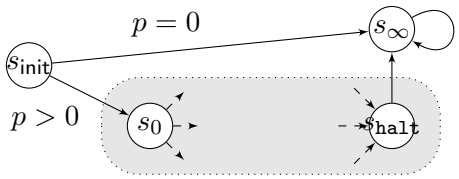


Figure 3: Encoding the halting problem into the language-preservation problem

Finally, notice that our reduction is readily adapted to the discrete-time setting, and/or to integer-valued parameters. □

Remark 3.3. *Our construction uses both p and $p-1$ in the clock constraints, as well as parametric constraints $p = 0$ and $p > 0$. This was not allowed in* [AHV93] *(where three different parameters were needed to compare the clocks with p, $p-1$ and $p+1$). Our construction could be adapted to only allow comparisons with $p-1$ (hence to use only one parameter), while keeping the number of clocks unchanged:*

- *the parametric constraints $p = 0$ and $p > 0$ could be respectively encoded as $(t = p) \wedge (t = 0)$ and $(t < p) \wedge (t = 0)$;*
- *transitions guarded by $x = p$ (which always reset the corresponding clock x) would then be encoded by a first transition with $x = p - 1$ resetting x and moving to a copy of $\mathcal{A}$ where we* remember *that the value of x should be shifted up by $p - 1$. All locations have invariant $x \leq 1$, and transitions guarded with $x = 1$, resetting x and returning to the main copy of $\mathcal{A}$. The same can be achieved for the other clocks, even if it means duplicating $\mathcal{A}$ several times (twice for each clock).*

Let us now show that this undecidability result is robust w.r.t. some variations in the definition of the problem.

Proposition 3.4. *Given a PTA $\mathcal{A}$ and a parameter valuation v, the existence of a valuation $v' \neq v$ such that the language of $v'(\mathcal{A})$ is a strict subset of that of $v(\mathcal{A})$ is undecidable (similarly for non-strict subset, and for strict and non-strict superset).*

Proof. We show that all four problems are undecidable:

($\subseteq$) The result follows directly from the encoding in Fig. 3: if the two-counter machine halts, then the untimed language is a^ω for some positive value of p, as well as for $p = 0$. If it does not halt, then the untimed language is made of finite words only when $p > 0$.

($\subsetneq$) Consider again the encoding in Fig. 3: add a transition from the initial location to a new location guarded with $p = 0$, and this time labeled with action b (recall that all other transitions are labeled with a).

With this new transition, the untimed language for $p = 0$ becomes $\{a^\omega, b\}$. Now, if the two-counter machine halts, for some positive value of the parameter, the untimed language of the automaton is $\{a^\omega\}$, which is a strict subset of the language of the automaton for $p = 0$. On the other hand, if the machine does not halt, then for any positive parameter valuation, the automaton reaches a deadlock, hence its untimed language is a (non-empty) set of finite words in a^+ (recall that a^+ denotes the set of all words made of an arbitrary number of as strictly greater than 0), and it is not a subset of $\{a^\omega, b\}$.

($\supseteq$) Same argument as for $\subseteq$.

($\supsetneq$) We use a reasoning dual to the $\subsetneq$ case: add a transition from the initial location to a new location guarded with $p > 0$, labeled with action b.

Then for $p = 0$, the untimed language still is $\{a^\omega\}$. Then if the two-counter machine halts, then for some positive value of p, the untimed language is $\{a^\omega, b\}$; if the machine does not halt, the language contains a finite word in a^+, whatever the (positive) value of p. □

We considered so far a definition of the untimed language as the set of untimed words associated with maximal runs, *i.e.* runs that are either infinite or blocking. An alternative definition of the untimed language could be the set of untimed words associated with all finite runs (non-necessarily maximal); note that this definition yields an untimed language that is prefix-closed. We prove that all results above (*i.e.* Theorem 3.1 and Proposition 3.4) extend to this alternative definition. We first consider the equality of language, and then the four variations of the problem, with (strict) inclusion instead of equality of the set of untimed words.

Proposition 3.5. *Given a PTA $\mathcal{A}$ and a parameter valuation v, the problems of the existence of a valuation $v' \neq v$ such that the set of non-necessarily maximal finite untimed words of $v'(\mathcal{A})$ is equal to, strictly included in, included in or equal to, larger than or equal to, or strictly larger than that of $v(\mathcal{A})$, respectively, are undecidable.*

Proof. We begin with handling untimed-language equality, again relying on the encoding in Fig. 3. For $p = 0$, the untimed language is a^+. For $p > 0$ if the two-counter machine does not halt, recall that any run will eventually reach a deadlock (for any positive value of p), yielding $a^{\leq N} = \{a^k \mid 0 \leq k \leq N\}$ (for some strictly positive $N \in \mathbb{N}$ depending on the value of p) as the untimed language. On the other hand, if the machine halts, then for some value of p, the automaton has an infinite run, and the untimed language is a^+.

This gives that there exists a parameter valuation $p > 0$ with the same set of non-necessarily maximal finite untimed words as for $p = 0$ if, and only if, the two-counter machine halts.

We now prove the results for inclusion relations:

($\supseteq$) The argument for language equality above also applies in this case.

($\subseteq$) We modify the construction to prove this case: from each main location s_i of the automaton, we add a transition labeled with b and guarded by $t = 0 \wedge z = p$: this transition can only be taken after p steps of the two-counter machine have been simulated. Then if the two-counter machine does not halt, for any positive value of p, the language contains at least one word that contains a b, hence it cannot be included in the language for $p = 0$; on the other hand, it the two-counter machine halts, then for some positive value of p both languages are the same (hence the inclusion holds).

($\subsetneq$) As for the same case in the proof of Prop. 3.4, it suffices to modify the automaton in order to add an extra word (*e.g.* b) when $p = 0$.

($\supseteq$) Similarly, it suffices to add one word to the language for any positive valuation of p. □

3.2. **Undecidability of the Trace Preservation Problem.** In this section, we provide two proofs of the following result:

Theorem 3.6. *The trace-preservation problem for PTA with one parameter is undecidable.*

We propose two different proofs of this result:

(1) the first proof (Section 3.2.1) is by a generic transformation of (parametric) timed automata without zero-delay cycle into one-location timed automata; the transformation involves diagonal constraints, uses an unbounded number of clocks, but does not increase the number of parametric clocks;
(2) the second proof (Section 3.2.2) does not involve diagonal constraints. It involves eight locations, but with an unbounded number of transitions and an unbounded number of parametric clocks.

3.2.1. *Encoding timed automata into one-location timed automata.* Our first proof relies on the encoding of TA (with the restriction that no sequence of more than k transitions may occur in zero delay, for some k; equivalently, those timed automata may not contain zero-delay cycles) into an equivalent TA with a single location; this reduction uses $k \times |L|$ additional clocks (where $|L|$ denotes the number of locations of $\mathcal{A}$) and requires diagonal constraints, *i.e.* constraints comparing clocks with each other (of the form $x_1 - x_2 \bowtie c$).

This result extends to PTA (provided that k does not depend on the value of the parameters), and the additional clocks are non-parametric. Using this reduction, the undecidability of the language preservation (Theorem 3.1) trivially extends to trace preservation. Let us first show the generic result for TA.

Proposition 3.7. *Let $\mathcal{A}$ be a TA in which any run starts with a positive delay, and such that for some k, no sequence of more than k transitions can occur in zero delay. Then there exists an equivalent TA $\mathcal{A}'$ with only one location and $k \times |\mathcal{A}| + 2$ additional clocks, such that the timed languages of $\mathcal{A}$ and $\mathcal{A}'$ are the same.*

Proof. We begin with the intuition behind our construction: each location ℓ of the automaton $\mathcal{A}$ is encoded using an extra clock x_ℓ, with the following property: when location ℓ is entered in the original automaton, the associated clock x_ℓ is reset in the one-location automaton. An extra clock x_0 is reset along each transition. This way, when the automaton is visiting ℓ, it holds $x_\ell - x_0 = 0$. However, the converse does not hold, because several transitions may be taken in zero delay.

To overcome this difficulty, we use $k+1$ copies of x_ℓ, numbered x_ℓ^0 to x_ℓ^k. In the encoding, each transition $t = (\ell, g, a, R, \ell')$ is encoded as several self-loops on the single location of $\mathcal{A}'$: for each $1 \leq i \leq k$, one self-loop encodes the effect of taking transition t as the i-th transition in a sequence of zero-delay transitions; additionally, one transition encodes the effect of taking t right after a positive delay. Formally:

- for each $1 \leq i \leq k$, one self-loop is guarded with the conjunction of g and

$$x_0 = 0 \wedge \left[x_\ell^{i-1} = 0 \wedge \bigwedge_{\ell'' \in L} x_{\ell''}^i > 0 \right];$$

  The first part of the latter constraint imposes that $x_0 = 0$, hence no delay may have elapsed since the previous transition. The second part of the constraint characterizes that a sequence of exactly i zero-delay transitions has been taken, and has reached location ℓ. In order to encode the effect of transition t, the self-loop carrying this guard is labeled with a, and resets the clocks in R and $x_{\ell'}^i$ (no need to reset x_0 as it is already zero).

- the last self-loop corresponds to transition t right after a positive delay; it is guarded with the conjunction of the guard g and of the constraint

$$x_0 > 0 \wedge \bigvee_{i \geq 0} \left[x_\ell^i - x_0 = 0 \wedge \bigwedge_{\ell'' \in L} x_{\ell''}^{i+1} - x_0 > 0 \right];$$

  Indeed, after a sequence of i zero-delay transitions (preceded by a non-zero-delay transition), it holds $x_\ell^{i+1} - x_0 > 0$ for all ℓ, and only the location ℓ reached at the end of the sequence satisfies $x_\ell^i = x_0$. This self-loop is labeled with a, and resets the clocks in R as well as x_0 and $x_{\ell'}^0$.

We require, for the time being, that initially all clocks have positive values, except for x_0 and $x_{\ell_0}^0$. In that case, there is a one-to-one correspondence between runs in $\mathcal{A}$ (never involving more than k consecutive transitions in zero delay) and those in $\mathcal{A}'$, so that both automata accept the same timed language.

Setting a special initial configuration is required for our encoding to be correct. The extra requirement that any run in $\mathcal{A}$ has to begin with a positive delay allows us to circumvent this problem: we add an extra clock x_1, which will never be reset (hence we have $x_0 = x_1$ only for the first transition); all transitions (ℓ_0, g, a, R, ℓ') from the initial location of the original automaton then give rise to a self-loop in the one-location automaton, guarded with $g \wedge x_0 > 0 \wedge x_0 = x_1$, labeled with a and resetting x_0, $x_{\ell'}^0$, and the clocks in R. □

The above transformation can obviously be applied to PTA, with the property that the timed language is preserved for any valuation of the parameters. Proposition 3.7 can be extended to PTA as follows:

Proposition 3.8. *Let $\mathcal{A}$ be a PTA for which there exists an integer k such that, for any parameter value, all runs start with a positive delay, and no sequence of more than k transitions occurs in zero delay. Then there exists an equivalent PTA $\mathcal{A}'$ with only one location and $k \times |\mathcal{A}| + 2$ additional clocks such that for any parameter valuation v, the timed languages of $v(\mathcal{A})$ and $v(\mathcal{A}')$ coincide.*

We slightly modify the PTA built in the proof of Theorem 3.1 (see Fig. 3) so that we can apply the transformation above: for this, we add a new location before s_{init} (so as to enforce a positive initial delay), and we constrain the self-loop on s_∞ so that some time elapses between consecutive occurrences.

Now, the resulting one-location PTA uses only one letter, so that its untimed language corresponds to its set of untimed traces. This proves our result.

As a remark, let us show that in the general case, deciding whether a given PTA contains no reachable zero-delay cycles, for some valuation of the parameters, is undecidable.

Theorem 3.9. *The existence of a parameter valuation v in a PTA $\mathcal{A}$ such that $v(\mathcal{A})$ contains no reachable zero-delay cycle is undecidable.*

Proof. Consider the two-counter machine encoding for the EF-emptiness problem in the proof of Theorem 3.1. It relies on the fact that the values of the counters are encoded modulo p, and that we can always find a big enough value of p to correctly encode an halting execution. We can therefore exclude values 0 and 1 from the possible values of p without changing anything in the proof. To do so we need only change the initial location to a fresh one and a transition form the new initial location to the former with guard $x = p \wedge x \geq 2$, that resets all the clocks.

Now, when $p \geq 2$, it is easily seen that all gadgets take at least 1 time unit to be traversed. So, if we add a self-loop with guard *true* to the location encoding the halting state, then there exists a parameter valuation v such that there is a reachable zero-delay cycle in $v(\mathcal{A})$ if, and only if, the two-counter machine halts. □

Knowing whether a PTA contains a zero-delay cycle for all parameter valuations is also undecidable: the construction above can be lifted to the undecidability proof for EF-universality found in [ALR16, Theorem 7].

3.2.2. *Proof with bounded number of locations.* We propose a second proof, where we avoid the use of diagonal constraints, at the expense of using unboundedly many parametric clocks. This proof follows the reduction of the proof of Theorem 3.1, but with only eight locations: one location is used to initialize the computation, and the other seven locations are then visited iteratively, in order to first update the information about the counters and then about the state of the two-counter machine. The location of the machine is then stored using as many clocks as the number of locations of the machine: the clock with least value (less than or equal to p) corresponds to the current location.

From a deterministic two-counter machine $\mathcal{M}$ with n states, we build a PTA with $n+4$ (parametric) clocks: n clocks q_1 to q_n are used to store the current location of $\mathcal{M}$ (the only clock with value less than or equal to the value of the parameter p corresponds to the current state of $\mathcal{M}$), two clocks x_1 and x_2 store the values of the two counters, clock t measures periods of p time units, and an extra clock r stores temporary information along the run. Intuitively, the PTA cycles between two main locations s_1 and t_1, each round in the cycle encoding the application of a transition of the two-counter machine (see Fig. 4): it goes from s_1 to t_1 for updating the values of the counters, and from t_1 back to s_2 for updating the clock encoding the new location of $\mathcal{M}$.

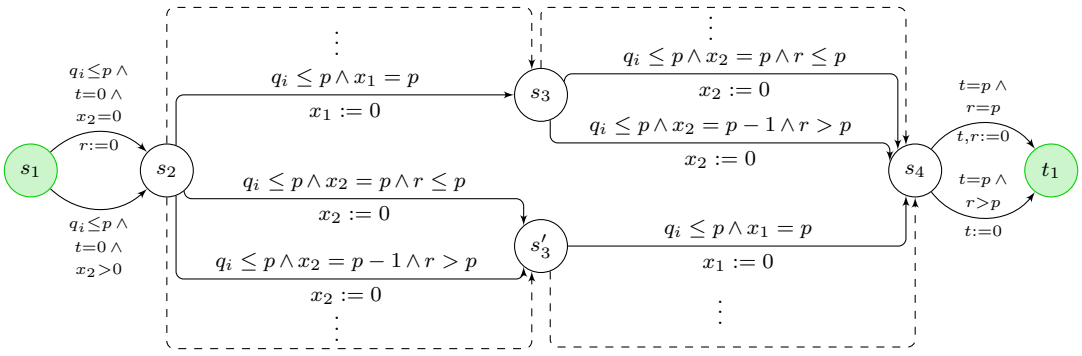


Figure 4: Encoding a two-counter machine (only one transition from q_i, testing and possibly decrementing clock x_2, has been encoded; the other transitions would add more transitions between s_2 and s_4). The module has two branches, going either through s_3 or s'_3, depending on the relative values of x_1 and x_2. Clock r is used to keep track whether the counter is zero.

More precisely, after spending $p+1$ time units in the initial location of the PTA (not displayed on Fig. 4), we take a transition resetting clocks q_1, x_1, x_2, r and t. This sets the initial configuration for starting the simulation of the two-counter machine. The PTA then

cycles between two modules. The first module (depicted on Fig. 4) is used to update the values of the clocks encoding the counters: depending on the instruction to perform (which only depends on the state of the two-counter machine, as it is deterministic), it first tests (between s_1 and s_2) whether the clock (x_1 or x_2) encoding the counter to be updated by the transition is zero, and resets clock r if needed (in order to remind that piece of information). It then updates the clock depending on the transition of $\mathcal{M}$. It also has to reset the other, non-updated clock when it reaches p; this may occur before or after the reset of the clock being updated, hence the two branches in the module. Finally, the module has a transition to its last location t_1, available when $t = p$.

From location t_1, a second module updates the values of clocks q_i depending on the transition to be performed in $\mathcal{M}$. It suffices to reset the clock q_j corresponding to the new location (while $t = 0$, and using the value of clocks q_i and r to get the next location to be visited; notice that this gives sufficient information since we assume that our two-counter machine is deterministic). The automaton then returns to s_1 after letting p time unit elapse, and resetting t and q_j whose values equal p.

This is a direct encoding of a two-counter machine as a PTA. It can easily be adapted to follow the reduction scheme of Theorem 3.1, which entails our result. Notice that by adding two extra clocks and two intermediary locations, we can get rid of comparisons with $p - 1$ and $p + 1$, in order to use only constraints of the form $x \sim p$ (see Remark 3.3).

3.3. **Undecidability of the Robust Language-Preservation Problem.** The robust language-preservation problem extends the discrete one by additionally requiring that the language is preserved on a "line" of valuations originating from the reference valuation. This is not the case of our previous proofs, which require a single parameter valuation for the reduction to be correct. In this section, we depart from the "discrete" setting of the previous section, and use rational-valued parameters and the full power of real-valued clocks.

Theorem 3.10. *The robust language-preservation problem for PTA with one (possibly bounded) parameter is undecidable.*

Proof. We begin by recalling from [ALR16] a reduction of the halting problem for counter machines to the EF-emptiness problem for 1-parameter PTA. The proof is then adapted to the language-preservation problem in the same way as for the proof of Theorem 3.6.

The encoding of the two-counter machine is as follows: it uses one rational-valued parameter p, one clock t to tick every time unit, and two parametric clocks x_1 and x_2 for storing the values of the counters c_1 and c_2, with $x_i = 1 - p \cdot c_i$ whenever $t = 0$.

An initial transition is used to initialize the values of x_1 and x_2 to 1, while it sets t to zero. It also checks that the value of p is in $(0, 1)$. Zero-tests are easily encoded by checking whether $x_i = 1$ while $t = 0$. Incrementation is achieved by resetting clock x_i when it reaches $1 + p$, while the other clocks are reset when they reach 1 (see Fig. 5). This way, exactly one time unit elapses in this module, and clock x_i is decreased by p, which corresponds to incrementing c_i. In Fig. 5, the upper branch corresponds to the case when $c_1 + 1 \leq c_2$ and the lower branch to the case when $c_2 \leq c_1 + 1$. When both values are equal, both branches can be taken, with the same effect. Decrementing is handled similarly. Finally, notice that the use of the constraint $x_i = 1 + p$ can be easily avoided, at the expense of an extra clock.

One easily proves that if a (deterministic) two-counter machine $\mathcal{M}$ halts, then by writing P for the maximal counter value reached during its finite computation, the PTA

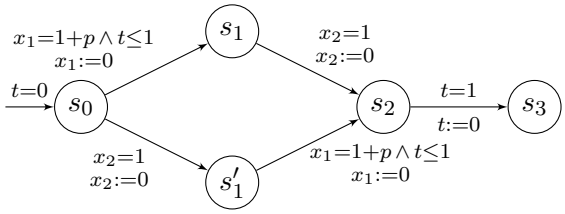


Figure 5: Encoding incrementation with a rational parameter

above has a path to the halting location as soon as $0 < p \leq 1/P$. Conversely, assume that the machine does not halt, and fix a parameter value $0 < p < 1$. If some counter of the machine eventually exceeds $1/p$, then at that moment in the corresponding execution in the associated PTA, the value of t when $x_i = 1 + p$ will be larger than 1, and the automaton will be in a deadlock. If the counters remain bounded below $1/p$, then the execution of the two-counter machine will be simulated correctly, and the halting state will not be reached.

We now adapt this construction to our language preservation problem. We have to forbid the infinite non-halting run mentioned above. For this, we add a third counter, which will be incremented every other step of the resulting three-counter machine, in the very same way as in the proof of Theorem 3.6. We then have the property that if $\mathcal{M}$ does not halt, the simulation in the associated PTA will be finite, for all non-zero parameter valuation. Adding locations s_{init} and s_∞ as in Fig. 3, we get the result that the two-counter machine $\mathcal{M}$ halts if, and only if, there is a parameter value $v_0(p) > 0$ such that all values $v(p)$ between 0 and $v_0(p)$ give rise to timed automata $v(\mathcal{A})$ accepting the same language as for $v_0(p) = 0$.

Finally, we notice that this reduction works even if we impose a positive upper bound on p (typically 1). □

3.4. **Undecidability of the Robust Trace Preservation Problem.** Combining Theorem 3.10 and the arguments of Section 3.2, we get:

Theorem 3.11. *The robust trace-preservation problem is undecidable for PTA with one (possibly bounded) parameter.*

Both proofs developed in Section 3.2 can be applied here:

- the first proof, using diagonal constraints (Section 3.2.1), applies as the PTA built above does not contain zero-delay cycles;
- the second proof (Section 3.2.2) also applies, by using one clock s_i per location ℓ_i of the two-counter machine with the encoding that the clock corresponding to the current location ℓ_i is the only clock s_i with value less than or equal to 1. Notice that we keep a bounded number of locations in that case.

3.5. **Undecidability over Bounded Time.** Let us now consider decision problems over bounded time, *i.e.* when the property must additionally be satisfied within T time units, for a given constant $T \geq 0$, and the system can thus be studied only inside that time frame. We first prove that the EF-emptiness problem is undecidable for PTA with three clocks and two rational-valued parameters, over bounded (dense) time. This result was already mentioned in [Jov13]; however, we provide here a full (and different) proof. Most importantly, this

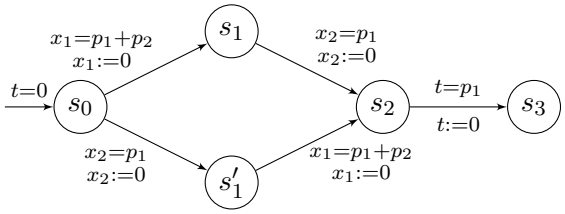


Figure 6: Encoding incrementation with rational parameters over bounded time

result will then be used to prove the undecidability of the problems considered earlier in this section in the time-bounded setting too.

Theorem 3.12. *The EF-emptiness problem is undecidable over bounded time for PTA with three clocks and two parameters.*

Proof. We reduce from the halting of a two-counter machine. Let us reuse the encoding of the proof of Theorem 3.10, and modify it as follows:

- The system is studied over 1 time unit (*i.e.* $T = 1$);
- We rename p into p_2;
- We replace any occurrence of "1" with a new parameter p_1; intuitively, this parameter will be small enough (compared to 1) to encode the length of the execution of the machine; in addition, p_1 must be sufficiently large when compared to p_2, so that p_2 can encode the maximum value of the counters. With our variables replacing, when $t = 0$, we now have the encoding $x_i = p_1 - p_2 \cdot c_i$. For any positive valuation of p_2, the maximum value of the counter that our encoding can support therefore becomes p_1/p_2.

We give the modified increment gadget in Fig. 6 and the decrement gadget in Fig. 7. The increment gadget requires p_1 time units to be traversed, and the decrement gadget requires $p_1 + p_2$ time units. The zero-test gadget (which requires 0 time unit in Theorem 3.10) is modified in an appropriate manner to require p_1 time units (see Fig. 8). Now, since any gadget requires at least p_1 time units, it is clear that, for any value of $p_1 > 0$, the number of operations that the machine can perform is finite, since the system executes over 1 time unit.

The initial gadget constrains p_1 to be strictly positive, and ensures that $x_1 = x_2 = 1$ while $t = 0$. In the gadget for incrementation, the upper branch corresponds to the case when $c_1 + 1 \leq c_2$ and the lower branch to $c_2 \leq c_1 + 1$. Similarly, in the decrementation gadget, the upper branch corresponds to $c_1 \leq c_2 + 1$ and the lower branch to $c_2 + 1 \leq c_1$. Finally, in the zero-test gadget, the upper branch corresponds to $c_1 \leq c_2$ and the lower branch to $c_2 \leq c_1$. In all these cases when both values are equal, both branches can be taken, with the same effect.

Let us prove that there exists a run reaching $s_{\texttt{halt}}$ in at most 1 time unit if, and only if, the two-counter machine halts.

(1) Assume that the machine does not halt. If $p_1 \leq 0$ then the initial gadget cannot be traversed. Now consider $p_1 > 0$. In this case, whatever the value of the parameters, after a maximum number of steps (at most $\frac{1}{p_1}$), one full time unit will elapse without the system reaching $s_{\texttt{halt}}$. In addition, if the value of p_2 is not small enough to encode the maximum value of the counters over these steps, an increment gadget will block, again

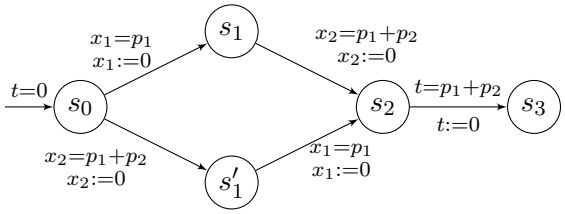


Figure 7: Encoding decrementation with rational parameters over bounded time

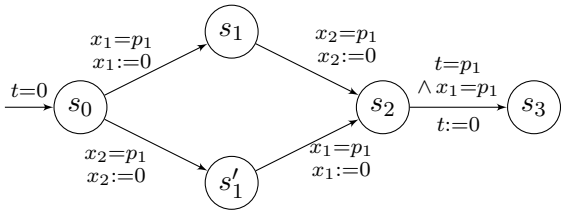


Figure 8: Encoding 0-test over bounded-time

without reaching $s_{\texttt{halt}}$. Hence if the two-counter machine does not halt, $s_{\texttt{halt}}$ cannot be reached within 1 time unit.

(2) Assume that the machine halts: in this case, if c is the maximum value of both C_1 and C_2 over the (necessarily finite) halting execution of the machine, and if m is the length of this execution, and if $c > 0$, then for valuations such that $p_2 \leq \frac{p_1}{c}$ and sufficiently small valuations of p_1 and p_2 (at most $p_1 + p_2 \leq \frac{1}{m}$ as any gadget takes at most $p_1 + p_2$ time units), then there exists one run that correctly simulates the machine, and eventually reaches $s_{\texttt{halt}}$. This set of valuations is non-empty: for example if $c > 0$, then we can choose $p_2 = \frac{1}{(c+1)m}$ and $p_1 = \frac{c}{(c+1)m}$ (since in the worst case, the sequence duration is $m(p_1+p_2)$); if $c = 0$ then $p_1 = p_2 = \frac{1}{m}$ (to allow for up to $\frac{m}{2}$ decrements or m zero-tests). Hence, if the two-counter machine halts, there exist parameter valuations for which a run reaches $s_{\texttt{halt}}$ within 1 time unit. □

Theorem 3.13. *The robust language-preservation problem for PTA with two (possibly bounded) parameters is undecidable over bounded time.*

Proof. By reusing the encoding of Theorem 3.12 in the reasoning of the proof of Theorem 3.10. □

Following a similar reasoning, we can also show the undecidability of all the other problems considered in this section in the time-bounded setting.

Remark 3.14. *Our undecidability results can be put into perspective with the decidability results for the larger class of hybrid automata of* [BDG+13]. *In* [BDG+13]*, time-bounded reachability is proved decidable for a subclass of hybrid automata with monotonic (either non-negative or non-positive) rates: parametric timed automata can fit into this framework: clocks and parameters all have non-negative rates (1 for clocks, and 0 for parameters), with the exception of the initialization phase: in that phase, we let time elapse until the parameters (growing at rate 1) reach their value, and then set their rates to 0; we then reset*

all clocks and start the real *execution of the automaton. However, to compare clocks and parameters together in a hybrid automaton, one needs diagonal constraints—which are not allowed in* [BDG+13]. *As we showed that our undecidability results (notably Theorem 3.12) hold over bounded-time with only two parameters, one can revisit the result of* [BDG+13] *as follows: allowing only two variables (our parameters) in diagonal constraints, with only two locations with a non-zero rate (the initialization locations for these parameters) makes the decidable problem of* [BDG+13] *undecidable.*

4. A Semi-Algorithm for the Trace Preservation Synthesis

In this section, we propose a semi-algorithm that solves the following *parameter-synthesis* problem: "*given a PTA $\mathcal{A}$ and a parameter valuation v, synthesize parameter valuations that yield the same language (or trace set) as v*".

The inverse method proposed in [ACEF09] outputs a parameter constraint that is a correct but incomplete answer to the trace-preservation problem. Below, we rewrite this algorithm so that, whenever it terminates, it outputs a correct answer for any PTA, and a complete answer for deterministic PTA.

4.1. The Algorithm TPSynth.

We give $\mathsf{TPSynth}(\mathcal{A}, v)$ in Algorithm 1. TPSynth maintains two constraints: K_{good} is the intersection of the parameter constraints associated with the v-compatible symbolic states met, whereas K_{bad} is the union[2] of the parameter constraints associated with all v-incompatible symbolic states. TPSynth also maintains two sets of symbolic states, *viz.* the set S of all symbolic states met, and the set S_{new} of symbolic states met at the latest iteration of the **while** loop. TPSynth is a breadth-first search algorithm, that iteratively explores the symbolic state space. Whenever a new symbolic state is met, its v-compatibility is checked (line 4). If it is v-compatible, its projection onto the parameters is added to K_{good} (line 4). Otherwise, its projection onto the parameters is added to K_{bad} (line 5), and the symbolic state is discarded from S_{new} (line 5), *i.e.* its successors will not be explored. When no new symbolic states can be explored, *i.e.* the set S_{new} is either empty or contains only symbolic states explored earlier (line 6), the intersection of v-compatible parametric constraints and the negation of the v-incompatible parametric constraints is returned (line 6). Otherwise, the algorithm explores one step further in depth (line 7). TPSynth is implemented in the IMITATOR software [AFKS12].

4.2. Soundness of TPSynth.

Theorem 4.1 states that, in case $\mathsf{TPSynth}(\mathcal{A}, v)$ terminates, its result is correct.

Theorem 4.1. *Let $\mathcal{A}$ be a PTA, let v be a parameter valuation. Assume $\mathsf{TPSynth}(\mathcal{A}, v)$ terminates with constraint K. Then $v \models \mathsf{K}$, and for all $v' \models \mathsf{K}$, $Traces(v'(\mathcal{A})) = Traces(v(\mathcal{A}))$.*

Let us prove Theorem 4.1 in the following. We first recall below a useful result stating that the projection onto the parameters of a constraint can only become more strict along a run.

Lemma 4.2. *Let $\mathcal{A}$ be a PTA, and let ρ be a run of $\mathcal{A}$ reaching (l, C). Then, for any successor (l', C') of (l, C), we have $C'{\downarrow_P} \subseteq C{\downarrow_P}$.*

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

Algorithm 1: $\mathsf{TPSynth}(\mathcal{A}, v)$

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

$\mathsf{K}_{good} \leftarrow \top$; $\mathsf{K}_{bad} \leftarrow \bot$; $S_{new} \leftarrow \{s_{\mathsf{init}}\}$; $S \leftarrow \varnothing$
while `true` **do**
  foreach *symbolic state* $(l, C) \in S_{new}$ **do**
    if $v \models C{\downarrow_P}$ **then** $\mathsf{K}_{good} \leftarrow \mathsf{K}_{good} \wedge C{\downarrow_P}$;
    else $\mathsf{K}_{bad} \leftarrow \mathsf{K}_{bad} \vee C{\downarrow_P}$; $S_{new} \leftarrow S_{new} \setminus \{(l, C)\}$;
  if $S_{new} \subseteq S$ **then return** $\mathsf{K}_{good} \wedge \neg \mathsf{K}_{bad}$;
  $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow \mathsf{Succ}(S_{new})$

Proof. Let (l, C) be a symbolic state. Let $e = (l, g, a, R, l')$ be a transition. Let (l', C') be the successor of (l, C) via e. Recall that, from the definition of the symbolic semantics, $C' = \big([(C \wedge g)]_R\big)^{\uparrow} \cap I(l')$.

Let $v \models C'{\downarrow_P}$. Then there exists $u' \models C'$ and a clock valuation w' such that $u'_{|P} = v$ and $u'_{|X} = w'$. From the definition of the Succ operator, there exists a clock valuation w, $u \models C$ and $d \geq 0$ such that $w' = w[R] + d$, $u_{|P} = v$ and $u_{|X} = w$. So $v \models C{\downarrow_P}$. □

We now recall below two results from [HRSV02].

Proposition 4.3. *Let $\mathcal{A}$ be a PTA, and let ρ be a run of $\mathcal{A}$ reaching (l, C). Let v be a parameter valuation. There exists an equivalent run in $v(\mathcal{A})$ if, and only if, $v \models C{\downarrow_P}$.*

Proof. From [HRSV02, Propositions 3.17 and 3.18]. □

Proposition 4.4. *Let $\mathcal{A}$ be a PTA, let v be a parameter valuation. Let ρ be a run of $v(\mathcal{A})$ reaching (l, w). Then there exists an equivalent symbolic run in $\mathcal{A}$ reaching (l, C), with $v \models C{\downarrow_P}$.*

Proof. From [HRSV02, Proposition 3.18]. □

Before proving Theorem 4.1, we need some intermediate results.

Lemma 4.5. *Let $\mathcal{A}$ be a PTA, let v be a parameter valuation. Assume $\mathsf{TPSynth}(\mathcal{A}, v)$ terminates with constraint K. Then $v \models \mathsf{K}$.*

Proof. By construction, all constraints added to K_{good} are v-compatible, hence their intersection is v-compatible. By construction, all constraints added to K_{bad} are v-incompatible, hence their union is v-incompatible; hence the negation of their union is v-compatible. This gives that $v \models \mathsf{K}_{good} \wedge \neg \mathsf{K}_{bad}$, thus $v \models \mathsf{K}$. □

Lemma 4.6. *Let $\mathcal{A}$ be a PTA, let v be a parameter valuation. Assume $\mathsf{TPSynth}(\mathcal{A}, v)$ terminates with constraint K. Then for all $v' \models \mathsf{K}$, we have $Traces(v'(\mathcal{A})) = Traces(v(\mathcal{A}))$.*

Proof. Let $v' \models \mathsf{K}$.

($\subseteq$) Let ρ' be a run of $v'(\mathcal{A})$, reaching a symbolic state (l, w'). From Proposition 4.4, there exists an equivalent run in $\mathcal{A}$ reaching a symbolic state (l, C'), with $v' \models C'{\downarrow_P}$.

We will now prove by *reductio ad absurdum* that $v \models C'{\downarrow_P}$. Assume $v \not\models C'{\downarrow_P}$. Hence (l, C') is either a v-incompatible symbolic state met in $\mathsf{TPSynth}(\mathcal{A}, v)$, or the successor of some v-incompatible symbolic state met in $\mathsf{TPSynth}(\mathcal{A}, v)$.

(1) Assume (l, C') is a v-incompatible symbolic state met in $\mathsf{TPSynth}(\mathcal{A}, v)$. By construction, $C'{\downarrow_P}$ has been added to K_{bad} (line 5 in Algorithm 1), hence $C'{\downarrow_P} \subseteq \mathsf{K}_{bad}$ hence $\neg\mathsf{K}_{bad} \cap C'{\downarrow_P} = \emptyset$ hence $(\mathsf{K}_{good} \wedge \neg\mathsf{K}_{bad}) \cap C'{\downarrow_P} = \emptyset$ hence $\mathsf{K} \cap C'{\downarrow_P} = \emptyset$. This contradicts that $v' \models \mathsf{K}$.
(2) Assume (l, C') is a v-incompatible symbolic state not met in $\mathsf{TPSynth}(\mathcal{A}, v)$, *i.e.* it belongs to some path starting from a v-incompatible symbolic state (l'', C'') met in $\mathsf{TPSynth}(\mathcal{A}, v)$. From Lemma 4.2, $C'{\downarrow_P} \subseteq C''{\downarrow_P}$, and hence $C'{\downarrow_P} \subseteq C''{\downarrow_P} \subseteq \mathsf{K}_{bad}$; then we apply the same reasoning as above to prove that $\mathsf{K} \cap C'{\downarrow_P} = \emptyset$, which contradicts that $v' \models \mathsf{K}$.

Hence $v \models C'{\downarrow_P}$. Now, from Proposition 4.3, there exists an equivalent run in $v(\mathcal{A})$, which gives that $\mathit{Traces}(v'(\mathcal{A})) \subseteq \mathit{Traces}(v(\mathcal{A}))$.

($\supseteq$) Let ρ be a run of $v(\mathcal{A})$, reaching a symbolic state (l, w). From Proposition 4.4, there exists an equivalent run in $\mathcal{A}$ reaching a symbolic state (l, C), with $v \models C{\downarrow_P}$. From the fixpoint condition of Algorithm 1, all v-compatible symbolic states of $\mathcal{A}$ have been explored in $\mathsf{TPSynth}(\mathcal{A}, v)$, hence $(l, C) \in S$, where S is the set of symbolic states explored just before termination of $\mathsf{TPSynth}(\mathcal{A}, v)$. By construction, $\mathsf{K} \subseteq C{\downarrow_P}$; since $v' \models \mathsf{K}$ then $v' \models C{\downarrow_P}$. Hence, from Proposition 4.3, there exists an equivalent run in $v'(\mathcal{A})$, which gives that $\mathit{Traces}(v'(\mathcal{A})) \supseteq \mathit{Traces}(v(\mathcal{A}))$. □

Theorem 4.1 immediately follows from Lemmas 4.5 and 4.6.

4.3. **Completeness of** TPSynth**.** We now state the completeness of TPSynth for *deterministic* PTA.

Theorem 4.7 (completeness of TPSynth)**.** *Let $\mathcal{A}$ be a deterministic PTA, let v be a parameter valuation. Assume $\mathsf{TPSynth}(\mathcal{A}, v)$ terminates with constraint K. Then $v' \models \mathsf{K}$ if, and only if, $\mathit{Traces}(v'(\mathcal{A})) = \mathit{Traces}(v(\mathcal{A}))$.*

Proof. Theorem 4.1 entails that $\mathit{Traces}(v'(\mathcal{A})) = \mathit{Traces}(v(\mathcal{A}))$ whenever $v' \models \mathsf{K}$. We prove the other implication. Let v' be a parameter valuation such that $\mathit{Traces}(v'(\mathcal{A})) = \mathit{Traces}(v(\mathcal{A}))$. The result comes from the fact that, in a deterministic (P)TA, the equality of trace sets implies the equivalence of runs. Hence we can prove a stronger result, that is $\mathsf{TPSynth}(\mathcal{A}, v) = \mathsf{TPSynth}(\mathcal{A}, v')$. Indeed, $\mathsf{TPSynth}(\mathcal{A}, v')$ proceeds by exploring symbolic states similarly to $\mathsf{TPSynth}(\mathcal{A}, v)$. From Proposition 4.3, the v-incompatible and v-compatible symbolic states met in $\mathsf{TPSynth}(\mathcal{A}, v')$ will be the same as in $\mathsf{TPSynth}(\mathcal{A}, v)$, and hence the constraints K_{good} and K_{bad} will be the same too. Hence $\mathsf{TPSynth}(\mathcal{A}, v) = \mathsf{TPSynth}(\mathcal{A}, v')$, which trivially gives that $v' \models \mathsf{TPSynth}(\mathcal{A}, v)$. □

Remark 4.8. *The incompleteness of* TPSynth *for nondeterministic PTA is easily seen: Consider the PTA $\mathcal{A}$ in Fig. 9. Clearly, the upper transition from l_0 to l_1 can only be taken if $p \leq 1$, and the lower transition if $p > 1$. Consider the valuation v assigning 0 to p. The (unique) trace in $v(\mathcal{A})$ is (l_0, a, l_1).*

Running $\mathsf{TPSynth}(\mathcal{A}, v)$, we get two symbolic states corresponding to l_1:

- *From the upper transition, we get $(l_1, x \geq 1 \wedge p \leq 1)$: this symbolic state is v-compatible; K_{good} is thus updated to the projection of this symbolic state onto P, i.e. $\mathsf{K}_{good} = p \leq 1$.*
- *From the lower transition, we get $(l_1, x \geq 1 \wedge p > 1)$: this symbolic state is v-incompatible, and therefore K_{bad} is updated to $p > 1$.*

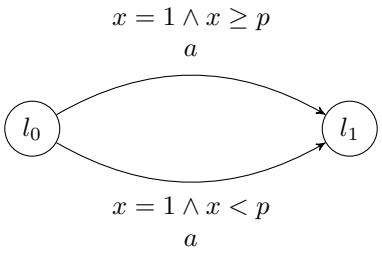


Figure 9: Non-deterministic PTA for which TPSynth is not complete

Eventually, $\mathsf{K}_{good} \wedge \neg \mathsf{K}_{bad}$ *is returned, that is* $p \leq 1 \wedge \neg(p > 1)$ *which gives* $p \leq 1$*. However, the trace* (l_0, a, l_1) *is in fact possible for* any *parameter valuation* $p \geq 0$*, and therefore the result output by* $\mathsf{TPSynth}(\mathcal{A}, v)$ *is not complete.*

5. Decidability Results for Subclasses of PTA

In this section, we first prove the finiteness of the parametric zone graph of 1-clock PTA over both discrete and rational time (Section 5.1). We then study the (un)decidability of the language and trace preservation emptiness problems for deterministic 1-clock PTA (Section 5.2), L/U-PTA (Section 5.3) and deterministic 1-parameter L-PTA and U-PTA (Section 5.4).

5.1. 1-Clock PTA.

In this section, we restrict the number of clocks of a PTA, without any restriction on the number of parameters. In fact, we even slightly extend the definition of PTA, by allowing parametric linear terms in guards and invariants.

Definition 5.1 (1-clock PTA)**.** *An* extended 1-clock PTA *(1cPTA for short) is a PTA with only one clock and possibly several parameters, and allowing guards and invariants of the form* $x \bowtie \sum_i \alpha_i p_i + c$*, with* $p_i \in P$ *and* $\alpha_i, c \in \mathbb{Z}$*.*

We show below that the parametric zone graph for 1cPTA is finite. In [AHV93], it is shown that the set of parameters for which there exists a run that can reach a given location can be computed for PTA over discrete time with only one parametric clock and arbitrarily many non-parametric clocks. Here, we lift the assumption of discrete time, we allow more general guards and invariants, and the finiteness of the parametric zone graph allows to synthesize valuations for more complex properties than pure reachability; however, we only consider a single (parametric) clock. Adding non-parametric clocks in this setting (perhaps reusing a construction used in [BBLS15]) is the subject of future work.

Definition 5.2. *Given a 1cPTA* $\mathcal{A}$*, a* 1-clock symbolic constraint *is a constraint over* $X \cup P$ *of the form* $\bigwedge_i (lt_i \bowtie x) \wedge \bigwedge_j (lt_j^1 \bowtie lt_j^2)$*, where* $i, j \in \mathbb{N}$*,* x *is the unique clock of* $\mathcal{A}$*, and* lt_i, lt_j^1, lt_j^2 *are parametric linear terms (i.e. of the form* $\sum_i \alpha_i p_i + c$*) either appearing in guards and invariants of* $\mathcal{A}$*, or equal to 0, and such that* lt_j^1, lt_j^2 *are all different from each other. We denote by* $\mathsf{1CSC}(\mathcal{A})$ *the set of 1-clock symbolic constraints of* $\mathcal{A}$*.*

Lemma 5.3. *Let $\mathcal{A}$ be a 1cPTA. Let (l, C) be a reachable symbolic state of $\mathcal{A}$. Then $C \in \mathsf{1CSC}(\mathcal{A})$.*

Proof. By induction on the length of the runs.

Base case: A run of length 0 consists of the sole initial symbolic state. According to the semantics of PTA, this symbolic state is $(l_{\mathsf{init}}, C_{\mathsf{init}})$, where C_{init} is $(X = 0)^{\uparrow} \wedge I(l_{\mathsf{init}})$, *i.e.* $x \geq 0 \wedge I(l_{\mathsf{init}})$. From Definition 5.1, $I(l_{\mathsf{init}})$ is of the form $\bigwedge_i lt_i \bowtie x$, with lt_i parametric linear terms of $\mathcal{A}$, hence $I(l_{\mathsf{init}}) \in \mathsf{1CSC}(\mathcal{A})$. Furthermore, $x \geq 0$ obviously belongs to $\mathsf{1CSC}(\mathcal{A})$. Hence the initial constraint belongs to $\mathsf{1CSC}(\mathcal{A})$.

Induction step: Consider a run of length n reaching the symbolic state (l, C), and assume C is of the form

$$\bigwedge_i (lt_i \bowtie x) \wedge \bigwedge_j (lt_j^1 \bowtie lt_j^2).$$

Let (l', C') be a successor of (l, C) through the Succ operation, for some edge (l, g, a, R, l'). Recall that $C' = \big([(C \wedge g)]_R\big)^{\uparrow} \cap I(l')$. Let us the consider the different operations sequentially.

Guard: From Definition 5.1, a guard is of the form $x \bowtie \sum_i \alpha_i p_i + c$, with $p_i \in P$ and $\alpha_i \in \mathbb{Z}$; hence $g \in \mathsf{1CSC}(\mathcal{A})$. Since $C \in \mathsf{1CSC}(\mathcal{A})$ by induction hypothesis, then $C \wedge g \in \mathsf{1CSC}(\mathcal{A})$.

Reset: Then, $[(C \wedge g)]_R$ is equivalent to removing x in $C \wedge g$ (using variable elimination technique such as Fourier-Motzkin) and adding a fresh equality $x = 0$. The elimination of x will leave the set of parametric inequalities (*i.e.* $\bigwedge_j lt_j^1 \bowtie lt_j^2$) unchanged. As for the inequalities containing x (*i.e.* $\bigwedge_i lt_i \bowtie x$), the elimination of x will lead to the disappearance of some of the lt_i, as well as the creation of new inequalities of the form $lt_i \bowtie lt_{i'}$, which will be added to the set of parametric inequalities (see, *e.g.* [Sch86]). Finally, adding $x = 0$ (which belongs to $\mathsf{1CSC}(\mathcal{A})$) makes $[(C \wedge g)]_R$ remain in $\mathsf{1CSC}(\mathcal{A})$.

Time elapsing: The time elapsing will remove some upper bounds on x, which leads to the disappearance of some of the inequalities, and hence makes $\big([(C \wedge g)]_R\big)^{\uparrow}$ still belong to $\mathsf{1CSC}(\mathcal{A})$.

Addition of the target invariant: The target invariant $I(l')$ adds new inequalities, all belonging to $\mathsf{1CSC}(\mathcal{A})$, hence $\big([(C \wedge g)]_R\big)^{\uparrow} \cap I(l') \in \mathsf{1CSC}(\mathcal{A})$.

Hence, $C' \in \mathsf{1CSC}(\mathcal{A})$. □

Theorem 5.4. *The parametric zone graph of a 1cPTA is finite.*

Proof. From Lemma 5.3, each symbolic state of a 1cPTA $\mathcal{A}$ belongs to $\mathsf{1CSC}(\mathcal{A})$. Due to the finite number of linear terms in the guards and invariants in $\mathcal{A}$ and the finite number of locations of $\mathcal{A}$, there is a finite number of possible symbolic states reachable in $\mathcal{A}$. □

Let us compute below an upper bound on the size of this symbolic graph. In the following, $|LT|$ denotes the number of different parametric linear terms (*i.e.* the number of guards and invariants) used in $\mathcal{A}$.

Proposition 5.5. *The parametric zone graph of a 1cPTA is in $|L| \times 2^{|LT|(|LT|+1)}$.*

Proof. First, note that, given a parametric linear term lt_i, an inequality $x \bowtie lt_i$ cannot be conjuncted with other $x \bowtie' lt_i$, where $\bowtie \neq \bowtie'$ (unless $\bowtie = \geq$ and $\bowtie' = \leq$ or the converse, in which case the conjunction is equivalent to a single equality). Hence, given lt_i, a 1-clock

symbolic constraint contains only one inequality of the form $x \bowtie lt_i$. The same reasoning applies to parametric inequalities $lt_j^1 \bowtie lt_j^2$.

There are $|LT|$ different linear terms in $\mathcal{A}$, and hence $|LT|$ different inequalities of the form $x \bowtie lt_i$ to be used in a 1-clock symbolic constraint. Following the same reasoning, there are $|LT|^2$ different inequalities of the form $lt_j^1 \bowtie lt_j^2$.

Hence the set $\mathsf{1CSC}(\mathcal{A})$ contains $2^{|LT|} \times 2^{|LT|^2} = 2^{|LT|(|LT|+1)}$ elements.

These constraints can be met for each of the $|L|$ locations. This gives that the zone graph of $\mathcal{A}$ contains at most $|L| \times 2^{|LT|(|LT|+1)}$ symbolic states. □

5.2. **Decidability and Synthesis for Deterministic 1-clock PTA.** We show here that the language- and trace-preservation problems are decidable for deterministic 1cPTA. These results rely on the correctness and completeness of Algorithm 1 and on the finiteness of the parametric zone graph of 1cPTA.

Theorem 5.6 (trace-preservation synthesis)**.** *Let $\mathcal{A}$ be a deterministic 1cPTA and v be a parameter valuation. The set of parameters for which the trace set is the same as in $v(\mathcal{A})$ is computable in time proportional to $|L| \times 2^{|LT|(|LT|+1)}$.*

Proof. Since $\mathcal{A}$ is a 1cPTA, then its parametric zone graph is finite from Theorem 5.4. Hence $\mathsf{TPSynth}(\mathcal{A}, v)$ terminates. Furthermore, since $\mathcal{A}$ is deterministic, from Theorems 4.1 and 4.7, $\mathsf{TPSynth}(\mathcal{A}, v)$ returns all parameter valuations v' such that $\mathit{Traces}(v'(\mathcal{A})) = \mathit{Traces}(v(\mathcal{A}))$.

Concerning the complexity, in the worst case, all symbolic states of $\mathcal{A}$ are v-compatible, and $\mathsf{TPSynth}(\mathcal{A}, v)$ needs to explore the entire parametric zone graph, which is of size $|L| \times 2^{|LT|(|LT|+1)}$. □

Theorem 5.7 (language-preservation synthesis)**.** *Let $\mathcal{A}$ be a deterministic 1cPTA and v be a parameter valuation. The set of parameters for which the language is the same as in $v(\mathcal{A})$ is computable in $|L| \times 2^{|LT|(|LT|+1)}$.*

Proof. Since $\mathcal{A}$ is deterministic, the set of parameter valuations v' such that $\mathit{Lang}(v'(\mathcal{A})) = \mathit{Lang}(v(\mathcal{A}))$ is the same as the set of parameter valuations v' such that $\mathit{Traces}(v'(\mathcal{A})) = \mathit{Traces}(v(\mathcal{A}))$. Hence one can directly apply $\mathsf{TPSynth}(\mathcal{A}, v)$ to compute the parameter valuations with the same language as $v(\mathcal{A})$. □

As direct corollaries of these results, the language- and trace-preservation problems are decidable for deterministic 1cPTA, and so are their *continuous* (robust) counterparts.

Remark 5.8. $\mathsf{TPSynth}$ *is not complete for non-deterministic 1cPTA: in fact, the PTA in Fig. 9 is a 1cPTA, and therefore Remark 4.8 applies here too.*

5.3. **Undecidability for L/U-PTA.** We showed so far that the language- and trace-preservation problems are undecidable for general PTA (Section 3) and decidable for (deterministic) 1-clock PTA (Section 5.2). These results match the EF-emptiness problem, also undecidable for general PTA [AHV93] and decidable for 1-clock PTA. We now show that the situation is different for L/U-PTA (PTA in which each parameter is always either used as a lower bound or always as an upper bound [HRSV02]): while EF-emptiness is decidable for L/U-PTA [HRSV02, BL09], we show that the language- and trace-preservation problems are not.

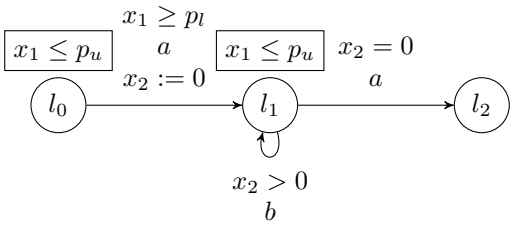


Figure 10: PTA gadget ensuring $p_l = p_u$

Constraining parameter equality. We first show how to encode equality of a lower-bound parameter and an upper-bound parameter in a L/U-PTA, using language preservation. Consider the PTA gadget depicted in Figure 10. Assume a parameter valuation v such that $p_l = p_u$. Note that since $p_l = p_u$, no time can elapse in l_1, and the b transition can never be taken. In fact, we have that the language of this gadget is aa iff $p_l = p_u$.

Now, one can rewrite the two-counter machine encoding of Section 3.1 using an L/U-PTA which, together with the previous gadget, gives the following undecidability result.

Theorem 5.9. *The language-preservation problem is undecidable for L/U-PTA with at least one lower-bound and at least one upper-bound parameter.*

First, the PTA gadget depicted in Fig. 10 can be characterized in the following lemma.

Lemma 5.10. *In the PTA gadget depicted in Figure 10, l_2 is reachable and b can never occur iff $p_l = p_u$.*

Proof. ($\Rightarrow$) Assume l_2 is reachable; hence, from the guards and invariants, we necessarily have $p_l \leq p_u$. Furthermore, b can occur iff it is possible to stay a non-null duration in l_1 iff $p_l < p_u$. Hence, b cannot occur implies $p_l \geq p_u$.
($\Leftarrow$) Assume $p_l = p_u$. Then no time can elapse in l_1, and hence b cannot occur. Furthermore, l_2 is obviously reachable for any such parameter valuation. □

We can now prove Theorem 5.9.

Proof. The proof is based on the reduction from the halting problem of a two-counter machine. The construction encodes the two-counter machine using an L/U-PTA with 2 parameters.

First, let us rewrite the two-counter machine encoding of Section 3.1 using L/U-PTA as follows. We split the parameter p used in the PTA $\mathcal{A}$ in the proof of Theorem 3.1 into two parameters p_l and p_u. Any occurrence of p as an upper-bound (resp. lower-bound) in a constraint is replaced with p_u (resp. p_l). Equalities of the form $p = x + c$ are replaced with $p_l \leq x + c \wedge p_u \geq x + c$.

Then, we plug the gadget in Figure 10 before the initial location of our modified encoding of the proof of Theorem 3.1; more precisely, we fuse l_2 in Fig. 10 with s_{init} in Fig. 3, and we reset all clocks in the transition from l_1 to l_2. This gives a new PTA, say $\mathcal{A}_{LU}$.

Let v be the reference parameter valuation such that $p_l = p_u = 0$. For v, the language of the gadget of Figure 10 is aa. Recall that in the proof of Theorem 3.1, the language of $p = 0$ is a^ω, and hence the language of our modified PTA $\mathcal{A}_{LU}$ for v is $aaa^\omega = a^\omega$.

(a) L-PTA (b) U-PTA

Figure 11: An L-PTA and a U-PTA for which the language differs for all $p \in \mathbb{N}$

Suppose the two-counter machine does not halt, and consider a parameter valuation $v' \neq v$. If $p_l \neq p_u$ in v', then from Lemma 5.10, the language of the gadget for v' is either a single deadlocked a (if $p_l > p_u$), or $ab^\omega|aa$ (if $p_l < p_u$); in both cases, the language of $v'(\mathcal{A}_{LU})$ differs from the language of $v(\mathcal{A}_{LU})$ (that is a^ω). If $p_l = p_u$, then we fall in the situation of Theorem 3.1: that is, there is no way for v' to accept the same language as v. Hence there exists no parameter valuation $v' \neq v$ such that the language is the same as for v.

Conversely, suppose the two-counter machine halts, and consider a parameter valuation $v' \neq v$. Again, if $p_l \neq p_u$ in v', then the language necessarily differs from v. If $p_l = p_u$, then we fall again in the situation of Section 3.1: for some $v' \neq v$ such that $p_l = p_u$ and p_l is large enough to encode the two counters maximum value, then the language is the same as for v. Hence there exists a parameter valuation $v' \neq v$ such that the language is the same as for v.

As a consequence, the two-counter machine halts iff there exists a parameter valuation $v' \neq v$ such that the language is the same as for v. □

This reasoning can be reused to prove the undecidability for L/U-PTA of the other problems considered in Section 3. It follows:

Theorem 5.11. (1) *The trace-preservation problem is undecidable for L/U-PTA with at least one lower-bound and at least one upper-bound parameter.*
(2) *The robust language- and trace-preservation problems are undecidable for L/U-PTA with at least one lower-bound and at least one upper-bound parameter.*

5.4. **A Decidability Result for 1-Parameter L-PTA and U-PTA.** In [BL09], a bound is exhibited for both L-PTA and U-PTA (*i.e.* PTA with only lower-bound, resp. upper-bound, parameters) such that either all parameter valuations beyond this threshold have an accepting run, or none of them has. This provides an algorithm for synthesizing all integer parameter valuations for which there exists an accepting run, by considering this bound, and then enumerate all (integer) valuations below this bound.

Unfortunately, such a bound for U-PTA (and L-PTA) does not exist for the language. Consider the U-PTA in Fig. 11b. Then, given $p \in \mathbb{N}$, the accepted language is $a^{\leq p}b^\omega$. Hence, it differs for all integer values of p. For L-PTA, the situation is similar: the language of the L-PTA in Fig. 11a is $a^p a^* b^\omega \cup a^\omega$, *i.e.* at least p times a followed (if the number of a is finite) by an infinite number of b.

We now show that the trace-preservation problem is decidable for deterministic L-PTA and U-PTA with a single integer parameter and arbitrarily many clocks: given a reference

integer parameter valuation v, it suffices to check $v+1$ and $v-1$ to decide whether another parameter valuation yields the same trace set as v.

Theorem 5.12. *The trace-preservation problem is* PSPACE*-complete for deterministic U-PTA and deterministic L-PTA with a single integer-valued parameter.*

Proof. Let $\mathcal{A}$ be a deterministic U-PTA with a single integer-valued parameter p (the reasoning is dual for L-PTA). Let v be a valuation of p. Construct the trace set of $v(\mathcal{A})$. Consider the valuation $v+1$ (*i.e.* the smallest integer valuation larger than v). It is known that increasing a parameter in a U-PTA can only *add* behaviors. Suppose $(v+1)(\mathcal{A})$ adds a behavior, *i.e.* enables a transition that was not enabled in $v(\mathcal{A})$. Since $\mathcal{A}$ is deterministic, then necessarily $(v+1)(\mathcal{A})$ contains a transition $l_1 \stackrel{a}{\Rightarrow} l_2$ that did not exist in $v(\mathcal{A})$. Hence the trace set of $(v+1)(\mathcal{A})$ strictly contains the trace set of $v(\mathcal{A})$, and the trace set of any valuation greater or equal to $v+1$ will again strictly contain the trace set of $v(\mathcal{A})$. Hence, deciding whether there exists a valuation greater than v for which the trace set is the same as $v(\mathcal{A})$ is equivalent to checking whether the trace set of $(v+1)(\mathcal{A})$ is the same as the trace set of $v(\mathcal{A})$.

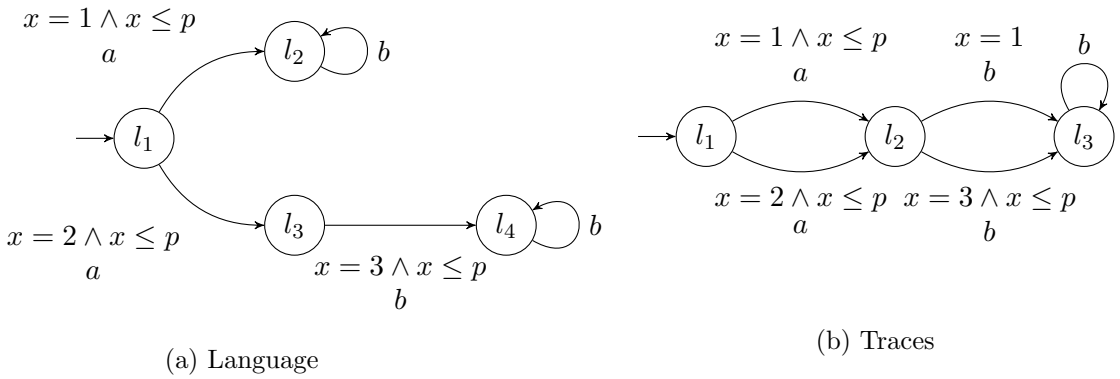
The proof for $v-1$ is symmetric. Hence it is decidable whether there exists a valuation different from v for which the trace set is the same as $v(\mathcal{A})$.

Now, for the PSPACE upper bound, we actually prove that testing the inclusion of the untimed language of timed automaton $\mathcal{A}_1$ in the untimed language of a *deterministic* timed automaton $\mathcal{A}_2$ can be done in PSPACE. Trace set inclusion can then be checked with untimed language inclusion. The proof is very similar to that of [AD94] for *timed language inclusion*.

Lemma 5.13. *The untimed language inclusion problem for deterministic timed automata is in* PSPACE*.*

Proof. We build a non-deterministic Turing machine that guesses a path in the product of the two automata. We store on the tape the current state, *i.e.* current location and region, as well as the next state and the action leading to it when they are non-deterministically guessed. We also need a counter for the maximum number of steps allowed for the path. When a new state and action are guessed, the machine verifies that the transition is indeed possible in both automata. If it is not possible in $\mathcal{A}_1$ then the machine rejects. If it is possible for $\mathcal{A}_1$ but not $\mathcal{A}_2$, or if the location is accepting in $\mathcal{A}_1$ but not in $\mathcal{A}_2$, we have found a witness for non-inclusion and the machine accepts. If it is possible for both automata, the machine overwrites the current state with the new state, increments the counter and proceeds to guessing a new successor, unless the counter has reached its maximum value, which is the product of the number of states of the region automata of $\mathcal{A}_1$ and $\mathcal{A}_2$. In this last case, the machine also rejects. Since, if an untimed word is accepted by $\mathcal{A}_1$ and not by $\mathcal{A}_2$, there must also be one such word with length less than the maximal value of the counter, it is clear that the machine accepts if and only if the untimed language of $\mathcal{A}_1$ is not included in that of $\mathcal{A}_2$. Finally, storing both states and actions can be done in polynomial space. As for the value of the counter, since its maximum value is exponential in the size of the problem, we need only a polynomial number of bits to store it in binary. So the procedure works in NPSPACE, and by Savitch's Theorem [Sav70], in PSPACE. □

Finally, PSPACE hardness is obtained by remarking that we can reduce reachability in timed automata to the trace preservation problem: Consider a deterministic timed automaton $\mathcal{A}$ (without parameter) and one of its location ℓ. Add a parameter p, a fresh



(a) Language

(b) Traces

Figure 12: Counterexamples for the method to decide the trace preservation emptiness in U-PTA

clock x, and a self-loop on ℓ, with guard $p \leq x \leq 0$. Finally for every transition arriving in ℓ, add a reset of x. The added transition is therefore possible only for $p = 0$. Then, it is clear that (i) the resulting PTA is an L-PTA with a single parameter that we can consider to be integer-valued, and (ii) there exists a value for p different from 0 with the same trace set as for $p = 0$ if and only if ℓ is not reachable in $\mathcal{A}$.

To get a U-PTA instead of an L-PTA, we can use guard $1 \leq x \leq p$. Again, we use the reference value $p = 0$ and the transition is this time not possible for $p = 0$ but it is for all other integer parameter values, so the result follows. □

Since we have a direct correspondence between trace sets and languages in deterministic automata, we get:

Theorem 5.14. *The language-preservation problem is* PSPACE*-complete for deterministic U-PTA and deterministic L-PTA with a single integer-valued parameter.*

Theorem 5.14 cannot be lifted to the language for non-deterministic L- and U-PTA. Consider the U-PTA in Fig. 12a: for $p = 1$, the language is ab^ω. For $p = 2$, the language is $ab^\omega|a$, which is different from $p = 1$. But then for $p \geq 3$, the language is again ab^ω. Hence testing only $v + 1 = 2$ is not enough, and the decidability in this case remains open.

Also note that these non-deterministic PTAs are *deterministic* with the usual definition of determinism (that two outgoing guards can have the same action label provided the guards are mutually disjoint—which is the case here with $x = 1$ and $x = 2$), which justifies our stronger definition of non-determinism.

Similarly, Theorem 5.12 cannot be lifted to the trace preservation in non-deterministic PTA, as witness in the U-PTA in Fig. 12b.

6. Conclusion and Perspectives

In this paper, we studied the decidability of the language- and trace-preservation problems in parametric timed automata. We summarize in Table 1 our (un)decidability results for PTA and its subclasses with arbitrarily many clocks; a red italicized cell denotes undecidability while a green plain cell denotes decidability. (1ip-dL&U-PTA stand for deterministic L-PTA,

resp. U-PTA, with one integer-valued parameter; L&U-PTA stand for L-PTA and U-PTA with rational-parameters or more than one clock or without our determinism assumption; bPTA stand for PTA with bounded parameters; d1cPTA stands for deterministic 1-clock PTA.) We also showed that both problems are decidable for deterministic PTA with a single clock. "N/A" indicates a problem that is not relevant for this class (robust versions of our problems are not so relevant for integer-valued parameters).

Future Works. First, we used an *ad-hoc* encoding of a 2-counter machine for our undecidability proofs, using four parametric clocks. In contrast, a new encoding of a 2-counter machine using PTA was proposed recently in [BBLS15] to show the undecidability of the EF-emptiness problem for integer-valued parameters, and that makes use of only three parametric clocks. It remains open whether the (non-robust) problems considered in our manuscript could be proved undecidable with as few as three parametric clocks in the case of integer-valued parameters. In addition, it remains to be proved whether the trace preservation problem is undecidable for a bounded number of clocks and without diagonal constraints.

Concerning the decidability for a single clock, it would be interesting to study whether this result can be adapted to PTA with one parametric clock and arbitrarily many non-parametric clocks, following the corner-point abstraction recently used in the construction of [BBLS15] to show the decidability of the EF-emptiness problem.

Language-preservation problems have been considered in [San11, AHJR12] in the setting of guard enlargement (for timed automata and time Petri nets): this is a weaker setting, in which the single parameter ε can only be used under the forms $x \geq a - \varepsilon$ and $x \leq b + \varepsilon$. This makes the robust version of the language-preservation problem decidable. In a similar flavor, time-abstract simulation of shrunk timed automata [SBM11] also shares commonalities with the problem we studied in the present paper. Identifying larger classes of PTA with decidable language-preservation problems in the light of these results is a relevant direction for future research.

Finally, we showed in [AL18] that some of the results presented in this paper extend to the smaller class of parametric event-recording automata [AL17b], *i.e.*, the language preservation problem remains undecidable in that setting. It remains however to prove whether the trace preservation problem is decidable for this subclass.

Acknowledgment

We are grateful to Olivier H. Roux and to the anonymous reviewers for useful comments.

Preservation	1ip-dL&U-PTA	L&U-PTA	bL/U-PTA	L/U-PTA	d1cPTA	bPTA	PTA
Language	Th. 5.14	open	*Th. 5.11*	*Th. 5.9*	Th. 5.7	*Th. 3.10*	*Th. 3.1*
Trace	Th. 5.12	open	*Th. 5.11*	*Th. 5.11*	Th. 5.6	*Th. 3.11*	*Th. 3.6*
Robust language	N/A	open	*Th. 5.11*	*Th. 5.11*	Th. 5.7	*Th. 3.10*	*Th. 3.10*
Robust trace	N/A	open	*Th. 5.11*	*Th. 5.11*	Th. 5.6	*Th. 3.11*	*Th. 3.11*

Table 1: Decidability of preservation emptiness problems for subclasses of PTA

References

[ACEF09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, October 2009.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In Dimitra Giannakopoulou and Dominique Méry, editors, *Proceedings of the 18th International Symposium on Formal Methods (FM 2012)*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, August 2012.

[AHJR12] S. Akshay, Loïc Hélouët, Claude Jard, and Pierre-Alain Reynier. Robustness of time Petri nets under guard enlargement. In Alain Finkel, Jérôme Leroux, and Igor Potapov, editors, *Proceedings of the 6th International Workshop on Reachability Problems (RP 2012)*, volume 7550 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2012.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing (STOC 1993)*, pages 592–601, New York, NY, USA, 1993. ACM.

[AL17a] Étienne André and Didier Lime. Liveness in L/U-parametric timed automata. In Alex Legay and Klaus Schneider, editors, *Proceedings of the 17th International Conference on Application of Concurrency to System Design (ACSD 2017)*, pages 9–18. IEEE, 2017.

[AL17b] Étienne André and Shang-Wei Lin. Learning-based compositional parameter synthesis for event-recording automata. In Ahmed Bouajjani and Silva Alexandra, editors, *Proceedings of the 37th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2017)*, volume 10321 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2017.

[AL18] Étienne André and Shang-Wei Lin. The language preservation problem is undecidable for parametric event-recording automata. *Information Processing Letters*, 136:17–20, 8 2018.

[ALR16] Étienne André, Didier Lime, and Olivier H. Roux. Decision problems for parametric timed automata. In Kazuhiro Ogata, Mark Lawford, and Shaoying Liu, editors, *Proceedings of the 18th International Conference on Formal Engineering Methods (ICFEM 2016)*, volume 10009 of *Lecture Notes in Computer Science*, pages 400–416. Springer, 2016.

[ALR18] Étienne André, Didier Lime, and Mathias Ramparison. TCTL model checking lower/upper-bound parametric timed automata without invariants. In David N. Jansen and Pavithra Prabhakar, editors, *Proceedings of the 16th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2018)*, volume 11022 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2018.

[AM15] Étienne André and Nicolas Markey. Language preservation problems in parametric timed automata. In Sriram Sankaranarayanan and Enrico Vicario, editors, *Proceedings of the 13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2015)*, volume 9268 of *Lecture Notes in Computer Science*, pages 27–43. Springer, September 2015.

[And19] Étienne André. What's decidable about parametric timed automata? volume 21, pages 203–219. Springer, 4 2019.

[BBLS15] Nikola Beneš, Peter Bezděk, Kim Gulstrand Larsen, and Jiří Srba. Language emptiness of continuous-time parametric timed automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015), Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 69–81. Springer, July 2015.

[BDG+13] Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joël Ouaknine, Jean-François Raskin, and James Worrell. Time-bounded reachability for monotonic hybrid automata: Complexity and fixed points. In Dang Van Hung and Mizuhito Ogawa, editors, *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA 2013)*, volume 8172 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2013.

[BL09] Laura Bozzelli and Salvatore La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009.

[BO14] Daniel Bundala and Joël Ouaknine. Advances in parametric real-time reasoning. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS 2014), Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2014.

[BY03] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

[Doy07] Laurent Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208–213, 2007.

[DWDMR08] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1-3):45–84, 2008.

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for timed automata. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015.

[Jov13] Aleksandra Jovanović. *Parametric Verification of Timed Systems*. PhD thesis, École Centrale de Nantes, 2013.

[Mil00] Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In Nancy A. Lynch and Bruce H. Krogh, editors, *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control (HSCC 2000)*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, Inc., 1967.

[San11] Ocan Sankur. Untimed language preservation in timed systems. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS 2011)*, volume 6907 of *Lecture Notes in Computer Science*, pages 556–567. Springer, August 2011.

[San13] Ocan Sankur. *Robustness in Timed Automata: Analysis, Synthesis, Implementation*. Thèse de doctorat, Laboratoire Spécification & Vérification, ENS Cachan, France, 2013.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, April 1970.

[SBM11] Ocan Sankur, Patricia Bouyer, and Nicolas Markey. Shrinking timed automata. In Supratik Chakraborty and Amit Kumar, editors, *Proceedings of the 31st Conferentce on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *Leibniz International Proceedings in Informatics*, pages 375–386. Leibniz-Zentrum für Informatik, December 2011.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.