Language Preservation Problems in Parametric Timed Automata

Étienne André and Nicolas Markey

June 2015

Research report LSV-15-05 (Version 1)

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Language Preservation Problems in Parametric Timed Automata*

Étienne André[1] and Nicolas Markey[2]

[1] Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, Villetaneuse, France
[2] LSV, CNRS & ENS Cachan, France

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 trace)?* We show that these problems are undecidable both for general PTA, and even for the restricted class of L/U-PTA. On the other hand, we exhibit decidable subclasses: 1-clock PTA, and 1-parameter deterministic L-PTA and U-PTA.

1 Introduction

Timed Automata. Timed Automata (TA hereafter) were introduced in the 1990's [1] 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 perfect 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 one of the integer constant in one of the clock constraints of this TA, does there exist another value of this constant 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 [7,11,12]. 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.

* This work is partially supported by the ANR national research program PACS (ANR-14-CE28-0002), and by European projects ERC EQualIS (308087) and FET Cassting (601148).

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* [3]) to compute a dense set of parameter valuations around a reference valuation v_0.

Parametric Timed Automata. In this paper, we tackle the language-preservation problem using *Parametric Timed Automata* (PTA) [2]. A PTA is a TA in which some of the constants in clock constraints are replaced by variables (a.k.a. parameters), whose value is not fixed 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 [2], for bounded rational valuations [10], etc. The proofs of these results exist in many different flavours, with various bounds on the number of parameters and clocks needed in the reductions.

To the best of our knowledge, the only non-trivial syntactic subclass of PTA with decidable EF-emptiness problem is the class of L/U-PTA [8]. 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 [8,6]. 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 [9].

Our Contributions. In this paper, we first prove that the language-preservation problem (and various related problems) is undecidable in most cases. 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.

Our second contribution is to 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 also study the decidability of these emptiness problems for subclasses of PTA: we prove decidability for PTA with a single clock, undecidability for L/U-PTA, and decidability for two subclasses of L/U-PTA with a single parameter.

A long version of this paper, with detailed proofs, is available as [4].

2 Definitions

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 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 $x \prec p + c$ or $x \prec c$ or $p \prec c$, where $\prec \in \{<, \leq, =, \geq, >\}$, $x \in X$, $p \in P$ 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 [2], where only atomic constraints of the form $x \prec p$ and $x \prec c$ (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 $x - x' \prec p + c$ or $x - x' \prec c$, where x and x' are two clocks and $\prec$, 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 1. We mainly focus here on continuous time (clock valuations take real values) and rational-valued parameters, as defined above. However, several of our results remain valid for discrete time (clock valuations take integer values) and integer-valued parameters. We will mention it explicitly when it is the case.

A valuation u satisfies an atomic constraint $\varphi\colon x \prec p + c$, which we denote $u \models \varphi$, whenever $u(x) \prec u(p) + c$. The definition for diagonal 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 $z \in \mathsf{dom}(u)$ 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 clock variables. This projection has the property that $v \models \Phi{\downarrow_P}$ if, and only if, there is an extension u of v to $X \cup P$ such that $u \models \Phi$. Such projections can be computed *e.g.* using 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 C by delaying an arbitrary amount of time. The time-elapsing of a constraint Φ is obtained by preserving all differences between any pair of clocks, preserving lower bounds, and relaxing upper bounds on atomic (single-clock) constraints. Given $R \subseteq X$, we define the *reset* of Φ, denoted by $[\Phi]_R$, as the constraint obtained from Φ by resetting the clocks in R, and keeping the other clocks unchanged. This is computed in the same way as projection above (*i.e.*, it corresponds to an existential quantification), and then adding constraints $x = 0$ on the clocks being reset.

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 [2].

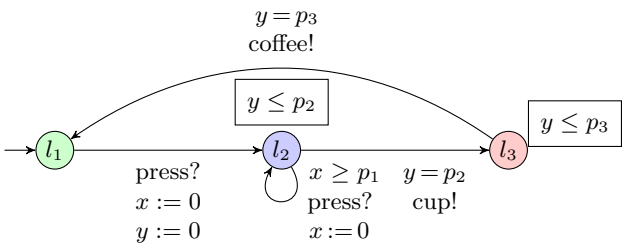


Fig. 1: An example of a coffee machine

Definition 2. *A* parametric timed automaton *(PTA for short) is a tuple* $\mathsf{A} = \langle \Sigma, L, l_{init}, X, P, I, \rightarrow \rangle$*, 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 set of clocks;* P *is a set of parameters;* I *assigns to every* $l \in L$ *a constraint* $I(l)$*, called the* invariant *of* l*;* $\rightarrow$ *is a set of edges* (l, g, a, R, l')*, also denoted by* $l \xrightarrow{g,a,R} l'$*, 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.*

For example, the PTA in Fig. 1 has 3 locations, 3 parameters p_1, p_2, p_3 and 2 clocks x, y.

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 $a' = a$.

Given a PTA $\mathsf{A} = \langle \Sigma, L, l_{\mathsf{init}}, X, P, I, \rightarrow \rangle$, and a parameter valuation v, $v(\mathsf{A})$ denotes the automaton obtained from A by substituting every occurrence of a parameter p_i by the constant $v(p_i)$ in the guards and invariants. Then $v(\mathsf{A})$ is a timed automaton [1], whose semantics is defined as follows:

Definition 3. *Given a PTA* $\mathsf{A} = \langle \Sigma, L, l_{init}, X, P, I, \rightarrow \rangle$*, and a parameter valuation* v*, the semantics of* $v(\mathsf{A})$ *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*$\}$*, with initial state* $q_{init} = (l_{init}, \mathbf{0}_X)$*, and* $((l, w), (d, e), (l', w')) \in \Rightarrow$ *whenever* e *is a transition* $(l, g, a, R, l') \in \rightarrow$ *such that* $(l, w + d) \models I(l) \wedge g$ *and* $w' = (w + d)[R \mapsto 0]$.

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 state 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 A, denoted by $Lang^t(\mathsf{A})$ (resp. $Lang(\mathsf{A})$), is the set of timed (resp. untimed) words associated with maximal 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 A is denoted by $Traces(\mathsf{A})$.

Given a state $s = (l, w)$, state s is said reachable in A under valuation v if s belongs to a run of $v(\mathsf{A})$; a location l is reachable if some state (l, w) is.

Following [8], we now define a symbolic semantics for PTA:

Definition 4 (Symbolic state). *A symbolic state of a PTA* 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 state $s = (l, \mathsf{C})$ is v-compatible if $v \models \mathsf{C}{\downarrow_P}$.

The computation of the state space relies on the Succ operation. The initial state of A is $s_{\mathsf{init}} = (l_{\mathsf{init}}, (X = 0)^{\uparrow} \wedge I(l_{\mathsf{init}}))$. Given a symbolic state $s = (l, \mathsf{C})$ and a transition $e = (l, g, a, R, l')$, we let $\mathsf{Succ}_e(s) = \{(l', \mathsf{C}') \mid \mathsf{C}' = \left([(\mathsf{C} \wedge g)]_R\right)^{\uparrow} \cap I(l')\}$ (notice that this is a singleton); we write $\mathsf{Succ}(s) = \bigcup_{e \in \rightarrow} \mathsf{Succ}_e(s)$. By extension, given a set S of 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 an infinite sequence of edges $(e_i)_i$ such that there exists a sequence of symbolic states $(s_i)_i$ such that $s_{i+1} = \mathsf{Succ}_{e_i}(s_i)$. 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.

In this paper, we address the following two problems:

Definition 5. *Given a PTA* 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* $Lang(v(\mathsf{A})) = Lang(v'(\mathsf{A}))$*;*
- *the* trace preservation problem *asks whether there exists another parameter valuation* v' *giving rise to the same set of traces (i.e. such that* $Traces(v(\mathsf{A})) = Traces(v'(\mathsf{A}))$ *[3].*

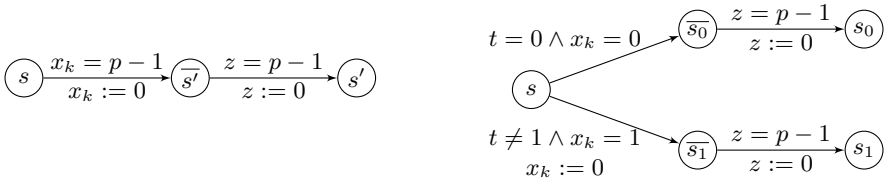
The continuous *versions of those problems additionally require that the language (resp. set of traces) is preserved under any intermediary valuation of the form* $\lambda \cdot v + (1 - \lambda) \cdot v'$*, for* $\lambda \in [0, 1]$ *(with the classical definition of addition and scalar multiplication).*

3 Undecidability of the Preservation Problems in General

3.1 Undecidability of the Language Preservation Problem

Theorem 6. *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 ("*does there exist a valuation of the parameters under which the target location is reachable?*"). We then extend the construction in order to prove the result.



(a) Incrementing c_k (b) Decrementing c_k

Fig. 2: Encoding a 2-counter machine

Fix a deterministic two-counter machine $\mathcal{M} = \langle S, T \rangle$. Our reduction requires 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 exactly when $t = 0$; 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: this is where our construction differs from the classical ones (e.g., [2,9]), as we use the parameter p to bound the length (number of step) of the possible halting computation of the two-counter machine. As the number of steps is bounded by p, we know that both c_1 and c_2 are also bounded by p. The parametric timed automaton A associated with $\mathcal{M}$ is defined as follows:

- its set of states has two copies of the set S of states of $\mathcal{M}$: for each $s \in S$, there is a *main state* with the same name s, and an *intermediary state* named $\overline{s}$;
- each state of A carries four self-loops, associated with each of the four clocks and reseting that clock when it reaches value p. This requires a global invariant enforcing the clocks t, x_1 and x_2 to remain below p, and clock z to remain below $p-1$.
  Then each transition (s, c_k++, s') incrementing counter c_k in $\mathcal{M}$ gives rise to a transition from state s to state $\overline{s'}$, with guard is $x_k = p-1$, and reseting clock x_k (see Figure 2a). Each transition of the form (s, c_k--, s_0, s_1) is handled similarly, but gives rise to two transitions: one transition from s to $\overline{s_0}$ with guard $t = 0 \wedge x_k = 0$, and one transition from s to $\overline{s_1}$ with guard $x_k = 1$ and reseting clock x_k. Then, from each state $\overline{s}$ of A, there is a transition to the corresponding state s with guard $z = p-1$ and reseting z (see Figure 2b).

This construction works as we expect (assuming p is an integer, which is easily checked by a simple initial module): clock t is reset every p time units (which cannot be seen in Figure 2 because we omitted the self-loops); clocks x_1 and x_2 keep track the values of c_1 and c_2, with the correspondence $x_k = c_k$ when $t = 0$; finally, clock z counts the number of steps (when considering the value of this clock when $t = 0$, it encodes a counter that is incremented at every transition of $\mathcal{M}$). Notice that clock z counts, but for the moment, it does not impose any constraint on the length of the simulation. Notice also that this construction currently does *not* correctly encode the runs of $\mathcal{M}$, since the counters are encoded modulo p.

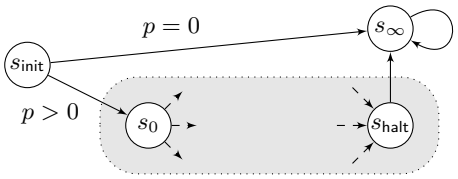


Fig. 3: Encoding the halting problem into the language-preservation problem

We modify our construction by adding the extra condition that $0 < t < p$ (or equivalently $1 \leq t \leq p-1$) to the guards $z = p-1$ of the transitions leaving the intermediary states. This way, when z (seen as a counter) has value $p-1$, no transition is available from any state s (or a transition to a sink state can be added), so that the encoding stops after mimicking $p-1$ steps of the execution of $\mathcal{M}$.

With this reduction, we have:

Lemma 7. *The two-counter machine $\mathcal{M}$ has a computation of length at most $p-1$ reaching s_{halt} from $(s_0, (c_1 = 0, c_2 = 0))$ if, and only if, there is a run reaching the corresponding state s_{halt} from $(s_0, (t = 0, x_1 = 0, x_2 = 0, z = 0))$ in $v(\mathsf{A})$.*

We now explain how to adapt this construction to the language preservation problem. The idea is depicted on Figure 3 (where all transitions are labeled with the same letter a): when $p = 0$, 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 $p > 0$, we have to enter the main part of the automaton A, and mimic the two-counter machine. From our construction above, the untimed language is the same if, and only if, the halting location is reachable.

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

Remark 8. 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 [2] (where three parameters were needed to compare the clocks with p, $p-1$ and $p+1$). Our construction could be adapted to only allow comparisons with $p-1$, while keeping the number of clocks unchanged:

- the parametric constraints $p = 0$ and $p > 0$ could be respectively encoded as $(x = p) \wedge (x = 0)$ and $(x < p) \wedge (x = 0)$;
- 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 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 A. The same can be achieved for the other clocks, even if it means duplicating A 16 times (twice for each clock).

3.2 Undecidability of the Trace Preservation Problem.

In this section, we provide two proofs of the following result:

Theorem 9. *The trace-preservation problem for PTA with one parameter is undecidable.*

The first proof is by a generic transformation of timed automata without zero-delay cycle into one-location timed automata; it involves diagonal constraints, but only a fixed number of parametric clocks. The second proof does not involve diagonal constraints, but it uses an unbounded number of parametric clocks.

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 clocks of A) and requires diagonal constraints, *i.e.* constraints comparing clocks with each other (of the form $x_1 - x_2 \prec c$).

This result extends to PTA, and the additional clocks are non-parametric. Using this reduction, the undecidability of the language preservation (Theorem 6) trivially extends to trace preservation. Let us first show the generic result for TA.

Proposition 10. *Let A be a TA in which, for some k, no sequence of more than k transitions occur in zero delay. Then there exists an equivalent TA A' with only one location and $k \times |\mathsf{A}| + 1$ additional clocks, such that the timed languages of A and A' are the same.*

Proof. We begin with the intuition behind our construction: each location ℓ of the automaton A is encoded using an extra clock x_ℓ, with the following property: when location ℓ is entered, the clock x_ℓ is reset. An extra clock x_0 is reset along each transition. Then 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^1 to x_ℓ^{k+1}. The exact encoding is then as follows: each transition (ℓ, g, a, R, ℓ') is encoded as several self-loops on the single location of A':

- one self-loop is guarded with the conjunction of the guard g and of the constraint $x_0 > 0 \wedge \bigvee_{i \geq 1} \left[x_\ell^i - x_0 = 0 \wedge \bigwedge_{\ell'' \in L} x_{\ell''}^{i+1} - x_0 > 0 \right]$; it is labeled with a, and resets the clocks in R as well as x^0 and $x_{\ell'}^1$.
- for each $1 \leq i \leq k$, one self-loop is guarded with the conjunction of g and $x_0 = 0 \wedge \left[x_\ell^i = 0 \wedge \bigwedge_{\ell'' \in L} x_{\ell''}^{i+1} > 0 \right]$; it is labeled with a and resets the clocks in R and $x_{\ell'}^{i+1}$.

With this transformation, we get a one-to-one correspondence between the run in A and in A', so that both automata have the same timed language. □

The above transformation can be applied to a PTA, with the property that the timed language is preserved for any valuation of the parameters. Proposition 10 can be extended to PTA as follows:

Proposition 11. *Let* A *be a PTA with no zero-delay cycle (for any valuation of the parameters). Then there exists an equivalent PTA* A' *with only one location and* $k \times |\mathsf{A}| + 1$ *additional clocks such that for any parameter valuation* v*, the timed languages of* $v(\mathsf{A})$ *and* $v(\mathsf{A}')$ *coincide.*

Now, for one-location automata, the untimed languages and the sets of untimed traces coincide. Applying this to our construction of Theorem 6 proves our result.

An Ad-Hoc Proof Avoiding Diagonal Constraints. 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 6, but with only four states: one state is used to initialize the computation, and the other three states are then visited cyclically, 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.

Formally, 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 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 (where p is the parameter), and an extra clock r stores temporary information along the run. Intuitively, the PTA cycles between two main states: it goes from the first one to the second one for updating the values of the counters, and from the second one back to the first one for updating clocks encoding the location of $\mathcal{M}$.

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 6, 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 $x \sim p$.

3.3 Undecidability of the Robust Language-Preservation Problem

The robust language-preservation problem extends 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 the parameter to take integer values 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 12. *The robust language preservation problem for PTA with one (possibly bounded) parameter is undecidable.*

Proof. We begin with a reduction[1] 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 9.

[1] This reduction for the EF problem we present here is an unpublished proof by Didier Lime; we develop the reduction here for our paper to be self-contained.

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 one parametric clock x_i for storing the value of each counter c_i, with $x_i = 1 - p \cdot c_i$ when $t = 0$.

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 reseting clock x_i when it reaches $1 + p$, while the other clocks are reset when they reach 1. This way, exactly one time unit elapses in this module, and clock x_i is decreased by a, which corresponds to incrementing c_i. 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 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 9. We then have the property that if $\mathcal{M}$ does not halt, the simulation in the associated PTA will be finite. Adding states 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 $v(\mathsf{A})$ accepting the same language. □

3.4 Undecidability of the Robust Trace Preservation Problem

Combining Theorem 12 and the arguments of Section 3.2, we get:

Theorem 13. *The robust trace-preservation problem is undecidable for PTA with one (possibly bounded) parameter.*

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* A *and a parameter valuation* v*, synthesize parameter valuations that yield the same language (or trace set) as* v".

The inverse method proposed in [3] outputs a parameter constraint that is a correct but non-complete 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.

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

input : PTA A, parameter valuation v
output : Constraint K over the parameters

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

We give $\mathsf{TPSynth}(\mathsf{A}, v)$ in Algorithm 1. TPSynth maintains two constraints: K_{good} is the intersection of v-compatible states met, whereas K_{bad} is the union[2] of all v-incompatible states. TPSynth also maintains two sets of states, *viz.* the set S of all states met, and the set S_{new} of 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 state is met, its v-compatibility is checked (line 4). If it is v-compatible, its projection onto the parameters is added to K_{good} (line 4). Otherwise, its projection onto the parameters is added to K_{bad} (line 5), and the state is discarded from S_{new} (line 5), *i.e.* its successors will not be explored. When no new states can be explored, *i.e.* the set S_{new} is either empty or contains only 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).

Theorem 14 states that, in case $\mathsf{TPSynth}(\mathsf{A}, v)$ terminates, its result is correct.

Theorem 14 (correctness of TPSynth). *Let* A *be a PTA, let* v *be a parameter valuation. Assume* $\mathsf{TPSynth}(\mathsf{A}, v)$ *terminates with constraint* K*. Then* $v \models \mathsf{K}$*, and for all* $v' \models \mathsf{K}$*,* $\mathit{Traces}(v'(\mathsf{A})) = \mathit{Traces}(v(\mathsf{A}))$*.*

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

Theorem 15 (completeness of TPSynth). *Let* A *be a deterministic PTA, let* v *be a parameter valuation. Assume* $\mathsf{TPSynth}(\mathsf{A}, v)$ *terminates with constraint* K*. Then* $v' \models \mathsf{K}$ *iff* $\mathit{Traces}(v'(\mathsf{A})) = \mathit{Traces}(v(\mathsf{A}))$*.*

Remark 16. The incompleteness of TPSynth for nondeterministic PTA is easily seen: consider a PTA with two states l and l', and two transitions from l to l' labeled with a and guarded with $x = p \wedge x \geq 5$ and $x = p \wedge x \leq 2$. Consider v such that $p = 0$. $\mathsf{TPSynth}(\mathsf{A}, v)$ outputs $p \leq 2$, whereas the complete set of parameter valuations with the same trace set as $v(\mathsf{A})$ is in fact $p \leq 2 \vee p \geq 5$.

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

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 17. *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* $x \prec \sum_i \alpha_i p_i + c$*, with* $p_i \in P$ *and* $\alpha_i \in \mathbb{Z}$*.*

We show below that the parametric zone graph for 1cPTA is finite. In [2], it is shown that the set of parameters reaching some 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. Eliminating non-parametric clocks in this setting is the subject of future work.

Definition 18. *Given a 1cPTA* A*, a* 1-clock symbolic constraint *is a constraint over* $X \cup P$ *of the form* $\bigwedge_i (lt_i \sim x) \wedge \bigwedge_j (lt_j^1 \sim lt_j^2)$*, where* $i, j \in \mathbb{N}$*,* x *is the unique clock of* A*, and* lt_i, lt_j^1, lt_j^2 *are parametric linear terms either (i.e. of the form* $\sum_i \alpha_i p_i + c$*) appearing in guards and invariants of* A*, or equal to 0, and such that* lt_j^1, lt_j^2 *are all different from each other. We denote by* $\mathsf{1CSC}(\mathsf{A})$ *the set of 1-clock symbolic constraints of* A*.*

Lemma 19. *Let* A *be a 1cPTA. Let* (l, C) *be a reachable symbolic state of* A*. Then* $\mathsf{C} \in \mathsf{1CSC}(\mathsf{A})$*.*

Proof sketch. We reason by induction on the length of the runs. For the base case, the initial state is obviously in $\mathsf{1CSC}(\mathsf{A})$. Then, for any state, we compute one of its successors using the Succ relation; and we show that each operation (intersection with the guard, resetting clocks, time elapsing, intersection with the invariant) makes the resulting constraint still belong to $\mathsf{1CSC}(\mathsf{A})$. □

Theorem 20. *The parametric zone graph of a 1cPTA is finite.*

Proof. From Lemma 19, each symbolic state of a 1cPTA A belongs to $\mathsf{1CSC}(\mathsf{A})$. Due to the finite number of linear terms in the guards and invariants in A and the finite number of locations of A, there is a finite number of possible symbolic states reachable in 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 A.

Proposition 21. *The parametric zone graph of a 1cPTA is in* $|L| \times 2^{|LT|(|LT|+1)}$.

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 22 (trace-preservation synthesis). *Let* 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* $v(\mathsf{A})$ *is computable in* $|L| \times 2^{|LT|(|LT|+1)}$.

Proof. Since A is a 1cPTA, then its parametric zone graph is finite from Theorem 20. Hence $\mathsf{TPSynth}(\mathsf{A}, v)$ terminates. Furthermore, since A is deterministic, from Theorems 14 and 15, $\mathsf{TPSynth}(\mathsf{A}, v)$ returns all parameter valuations v' such that $\mathit{Traces}(v'(\mathsf{A})) = \mathit{Traces}(v(\mathsf{A}))$.

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

Theorem 23 (language-preservation synthesis). *Let* A *be a deterministic 1cPTA and* v *be a parameter valuation. The set of parameters for which the language is the same as in* $v(\mathsf{A})$ *is computable in* $|L| \times 2^{|LT|(|LT|+1)}$.

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

As direct corollaries of these results, the language- and trace-preservation problems are decidable for deterministic 1cPTA.

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 [2] and decidable for 1-clock PTA (at least over discrete time). 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 [8]): while EF-emptiness is decidable for L/U-PTA [8,6], we show that the language- and trace-preservation problems are not.

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 4. 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 $p_l = p_u$.

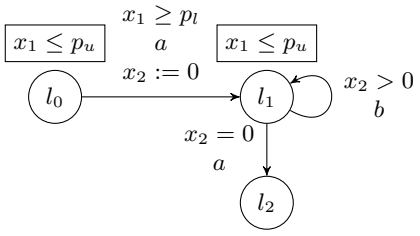


Fig. 4: PTA gadget ensuring $p_l = p_u$

Now, one can rewrite the 2CM encoding of Section 3.1 using an L/U-PTA which, together with the previous gadget, gives the following undecidability result.

Theorem 24. *The language-preservation problem is undecidable for L/U-PTA with at least one lower-bound and at least one upper-bound parameter.*

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

Theorem 25. *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 [6], 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, as witnessed by the U-PTA of Fig. 5: given $p \in \mathbb{N}$, the accepted language is $a^{\leq p}b^{\omega}$. A similar L-PTA example is easily obtained.

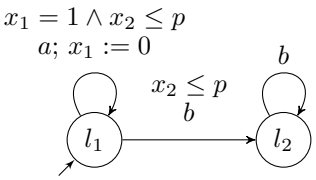


Fig. 5: An U-PTA with different language for each parameter valuation

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 26. *The trace-preservation problem is decidable for deterministic U-PTA and deterministic L-PTA with a single integer-valued parameter.*

Proof. Let 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 $v(\mathsf{A})$. 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 $v+1(\mathsf{A})$ adds a behavior, *i.e.* enables a transition that was not enabled in $v(\mathsf{A})$. Since A is deterministic, then necessarily $v+1(\mathsf{A})$ contains a transition $l_1 \stackrel{a}{\Rightarrow} l_2$ that did not exist in $v(\mathsf{A})$. Hence the trace set of $v+1(\mathsf{A})$ strictly contains the trace set of $v(\mathsf{A})$, and the trace set of any valuation greater or equal to $v+1$ will again strictly contain the trace set of $v(\mathsf{A})$. Hence, deciding whether there exists a valuation greater than v for which the trace set is the same as $v(\mathsf{A})$ is equivalent to checking whether the trace set of $v+1(\mathsf{A})$ is the same as the trace set of $v(\mathsf{A})$.

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 $v(\mathsf{A})$. □

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

Theorem 27. *The language-preservation problem is decidable for deterministic U-PTA and deterministic L-PTA with a single integer-valued parameter.*

Theorem 27 cannot be lifted to the language for non-deterministic L- and U-PTA. Consider the U-PTA in Fig. 6: 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.

Similarly, a counter-example to the extension of Theorem 26 to non-deterministic L- and U-PTA can be obtained easily.

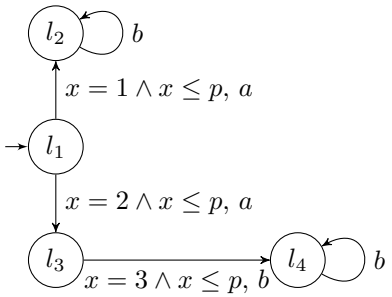


Fig. 6

6 Conclusion and Perspectives

In this paper, we studied the decidability of the language and trace preservation emptiness. We summarize in Table 1 our (un)decidability results for PTA and its subclasses with arbitrarily many clocks; an italicized cell denotes undecidability. (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; bPTA stand for PTA with bounded parameters.) We also showed that both problems are decidable for deterministic PTA with a single clock.

Future Works. First, we used an *ad-hoc* encoding of a 2-counter machine for our proofs of undecidability, using four parametric clocks. In contrast, a new encoding of a 2-counter machine using PTA was proposed very recently in [5],

Preservation	1ip-dL&U-PTA	L&U-PTA	bL/U-PTA	L/U-PTA	bPTA	PTA
Language	Th. 27	open	*Th. 25*	*Th. 24*	*Th. 12*	*Th. 6*
Trace	Th. 26	open	*Th. 25*	*Th. 25*	*Th. 13*	*Th. 9*
Robust language	open	open	*Th. 25*	*Th. 25*	*Th. 12*	*Th. 12*
Robust trace	open	open	*Th. 25*	*Th. 25*	*Th. 13*	*Th. 13*

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

that makes use of only three parametric clocks. We assume that our proofs could be rewritten using that encoding, proving the undecidability of the problems considered in this paper with as few as three clocks.

A promising direction to find decidability results consists in considering L-PTA and U-PTA. Furthermore, our results are linked to the robustness of timed systems; future works consist in finding the boundary between expressive models of robustness (with many parameter dimensions), that are undecidable, and less expressive models (usually with a single parameter), that are decidable.

Acknowledgement. We thank Didier Lime for telling us about the reduction we used in the proof of Theorem 12.

References

[1] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *ICALP*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

[2] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM Press, 1993.

[3] É. André, Th. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *IJFCS*, 20(5):819–836, 2009.

[4] É. André and N. Markey. Language preservation problems in parametric timed automata. Research Report LSV-15-05, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2015.

[5] N. Beneš, P. Bezděk, K. G. Larsen, and J. Srba. Language emptiness of continuous-time parametric timed automata. In *ICALP, Part II*, volume 9135 of *LNCS*. Springer, July 2015. To appear.

[6] L. Bozzelli and S. La Torre. Decision problems for lower/upper bound parametric timed automata. *FMSD*, 35(2):121–151, 2009.

[7] M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robust safety of timed automata. *FMSD*, 33(1-3):45–84, 2008.

[8] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

[9] A. Jovanović, D. Lime, and O. H. Roux. Integer parameter synthesis for timed automata. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015.

[10] J. S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *HSCC*, volume 1790 of *LNCS*, pages 296–309, 2000.

[11] O. Sankur. Untimed language preservation in timed systems. In *MFCS*, volume 6907 of *LNCS*, pages 556–567. Springer, 2011.

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

A Proofs of Section 3

Theorem 6. *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 ("*does there exist a valuation of the parameters under which the target location is reachable?*"). We then extend the construction in order to prove the result.

Fix a deterministic two-counter machine $\mathcal{M} = \langle S, T \rangle$. Our reduction requires 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 exactly when $t = 0$; 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: this is where our construction differs from the classical ones (e.g., [2,9]), as we use the parameter p to bound the length (number of step) of the possible halting computation of the two-counter machine. As the number of steps is bounded by p, we know that both c_1 and c_2 are also bounded by p. The parametric timed automaton A associated with $\mathcal{M}$ is defined as follows:

- its set of states contains exactly two copies of the set S of states of $\mathcal{M}$: for each $s \in S$, there is a *main state* with the same name s, and an *intermediary state* named $\overline{s}$;
- each state of A carries four self-loops, associated with each of the four clocks and reseting that clock when it reaches value p. This requires a global invariant enforcing the clocks t, x_1 and x_2 to remain below p, and clock z to remain below $p - 1$.
  Then each transition $(s, c_k + +, s')$ incrementing counter c_k in $\mathcal{M}$ gives rise to a transition from state s to state $\overline{s'}$, with guard is $x_k = p - 1$, and reseting clock x_k (see Figure 2a). Each transition of the form $(s, c_k - -, s_0, s_1)$ is handled similarly, but gives rise to two transitions: one transition from s to $\overline{s_0}$ with guard $t = 0 \wedge x_k = 0$, and one transition from s to $\overline{s_1}$ with guard $x_k = 1$ and reseting clock x_k. Then, from each state $\overline{s}$ of A, there is a transition to the corresponding state s with guard $z = p - 1$ and reseting z (see Figure 2b).

This construction works as we expect (assuming p is an integer, which is easily checked by a simple initial module): clock t is reset every p time units (which cannot be seen in Figure 2 because we omitted the self-loops); clocks x_1 and x_2 keep track the values of c_1 and c_2, with the correspondence $x_k = c_k$ when $t = 0$; finally, clock z counts the number of steps (when considering the value of this clock when $t = 0$, it encodes a counter that is incremented at every transition of $\mathcal{M}$). Notice that clock z counts, but for the moment, it does not impose any

constraint on the length of the simulation. Notice also that this construction currently does *not* correctly encode the runs of $\mathcal{M}$, since the counters are encoded modulo p.

We modify our construction by adding the extra condition that $0 < t < p$ (or equivalently $1 \leq t \leq p-1$) to the guards $z = p-1$ of the transitions leaving the intermediary states. This way, when z (seen as a counter) has value $p-1$, no transition is available from any state s (or a transition to a sink state can be added), so that the encoding stops after mimicking $p-1$ steps of the execution of $\mathcal{M}$.

Now, fix an integer value v for p. With a run of $v(\mathsf{A})$, we associate a sequence built as follows: with each occurrence of a transition from some s to $\overline{s}$, we associate a configuration of $\mathcal{M}$. If the transition enters state $\overline{s}$ with clock valuation $(x_1, x_2, z = 0, t)$, we let $m = v(p) - t$, $\ell_m = s$, and $c_i^m = x_i + m - v(p)$. We claim that (ℓ_m, c_1^m, c_2^m) is the m-th configuration of the (unique) run of $\mathcal{M}$. We prove this result inductively: the first time the transition is taken, we reach $\overline{s}_0$ with $x_1 = x_2 = t = v(p) - 1$, so that $m = 1$, $\ell_1 = s_0$, and $c_1^1 = c_2^1 = 0$, which indeed corresponds to the first configuration of any run of $\mathcal{M}$. Now, if the m-th configuration is (ℓ_m, c_1^m, c_2^m) and corresponds to entering state $\overline{\ell}_m$ with clock valuation $x_1 = v(p) - m + c_1^m$, $x_2 = v(p) - m + c_2^m$, $t = v(p) - m$ and $z = 0$ (with the additional properties that $c_i^m < m$), we consider three possible evolutions of $\mathcal{M}$:

- if ℓ_m increments counter c_k, then a delay of $m - c_k^m - 1$ is elapsed in $\overline{\ell}_m$, and then the transition to some s' is taken, with clock x_k being reset. In s', a delay of $v(p) - m + c_k^m$ is elapsed before going to $\overline{s}'$. In the end, a total delay of $v(p) - 1$ has elapsed, but t, x_1 and x_2 have been reset when they did reach $v(p)$, which equivalently amounts to decreasing their values by $v(p)$. So the new value of t is one less than the value when entering $\overline{\ell}_m$, which means that m has been incremented. Similarly, we can check that the values of the counters have been correctly updated.
- the other cases (zero test and possible decrement) are handled similarly.

It follows:

Lemma 7. *The two-counter machine $\mathcal{M}$ has a computation of length at most $p-1$ reaching s_{halt} from $(s_0, (c_1 = 0, c_2 = 0))$ if, and only if, there is a run reaching the corresponding state s_{halt} from $(s_0, (t = 0, x_1 = 0, x_2 = 0, z = 0))$ in $v(\mathsf{A})$.*

We now explain how to adapt this construction to the language preservation problem. The idea is depicted on Figure 3 (where all transitions are labeled with the same letter a): when $p = 0$, 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 $p > 0$, we have to enter the main part of the automaton A, and mimic the two-counter machine. From our construction above, the untimed language is the same if, and only if, the halting location is reachable.

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

Theorem 9. *The trace-preservation problem for PTA with one parameter is undecidable.*

Encoding timed automata into one-location timed automata.

Proposition 10. *Let* A *be a TA in which, for some* k*, no sequence of more than* k *transitions occur in zero delay. Then there exists an equivalent TA* A' *with only one location and* $k \times |\mathsf{A}| + 1$ *additional clocks, such that the timed languages of* A *and* A' *are the same.*

Proof. We begin with the intuition behind our construction: each location ℓ of the automaton A is encoded using an extra clock x_ℓ, with the following property: when location ℓ is entered, the clock x_ℓ is reset. An extra clock x_0 is reset along each transition. We then have the property that when we are in 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^1 to x_ℓ^{k+1}. The exact encoding is then as follows: each transition (ℓ, g, a, R, ℓ') is encoded as several self-loops on the single location of A':

- one self-loop is guarded with the conjunction of g, $x_0 > 0$, and

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

  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 state ℓ reached at the end of the sequence satisfies $x_\ell^i = x_0$. The self-loop is labeled with a, and resets the clocks in R as well as x^0 and $x_{\ell'}^1$.
- for each $1 \leq i \leq k$, one self-loop is guarded with the conjunction of g, $x_0 = 0$, and

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

  it is labeled with a and resets the clocks in R and $x_{\ell'}^{i+1}$.

With this transformation, we get a one-to-one correspondence between the run in A and the runs in A', so that both automata have the same timed language. □

An Ad-Hoc Proof Avoiding Diagonal Constraints. 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 6, but with only four states: one state is used to initialize the computation, and the other three states are then visited cyclically, 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

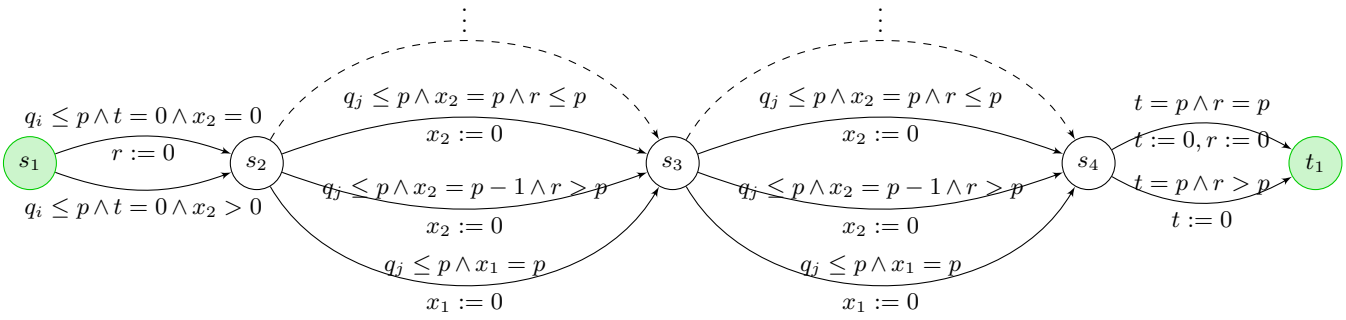


Fig. 7: Encoding a two-counter machine (only a transition from q_i testing and decrementing clock x_2 has been encoded; the other transitions would add more transitions between s_2 and s_4).

clocks as the number of locations of the machine: the clock with value less than or equal to one corresponds to the current location.

Formally, 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 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 (where p is the parameter), and an extra clock r stores temporary information along the run. Intuitively, the PTA cycles between two main states: it goes from the first one to the second one for updating the values of the counters, and from the second one back to the first one for updating clocks encoding the location of $\mathcal{M}$.

More preciely, after spending p time units in the initial location of the PTA, we take a transition resetting clocks q_1, x_1, x_2, z and t. The run of the PTA then successively visits two modules. The first module (see Fig. 7) is used to update the values of the clocks encoding the counters: depending on the instruction to perform, it first tests whether clock x_1 or x_2 is zero, and resets clock r if needed (in case that counter is tested to zero and actually is zero). It then performs the operation on x_1 or x_2, depending on the current state of $\mathcal{M}$. It also resets the non-updated clock when it reaches p. Finally, it waits until $t = p$.

From state 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 corresponding to the new location (while $t = 0$, and using the value of clock r to decide the issue of the possible zero test). The automaton then returns to s_1 after letting p time unit elapse, and resetting t and the clock q_i whose value equals 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 6, 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 $x \sim p$.

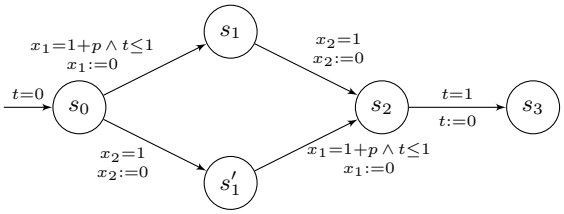


Fig. 8: Encoding incrementation with a rational parameter

Theorem 12. *The robust language preservation problem for PTA with one (possibly bounded) parameter is undecidable.*

Proof. We begin with a reduction[3] 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 9.

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 one parametric clock x_i for storing the value of each counter c_i, with $x_i = 1 - p \cdot c_i$ when $t = 0$.

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 reseting clock x_i when it reaches $1 + a$, while the other clocks are reset when they reach 1 (see Fig. 8). This way, exactly one time unit elapses in this module, and clock x_i is decreased by a, which corresponds to incrementing c_i. 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 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 9. We then have the property that if $\mathcal{M}$ does not halt, the simulation in the associated PTA will be finite. Adding states 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 $v_0(p) > 0$

[3] This reduction for the EF problem we present here is an unpublished proof by Didier Lime; we develop the reduction here for our paper to be self-contained.

such that all values $v(p)$ between 0 and $v_0(p)$ give rise to timed automata $v(\mathsf{A})$ accepting the same language. □

Theorem 13. *The robust trace-preservation problem is undecidable for PTA with one (possibly bounded) parameter.*

The two different proofs we developed at Section 3.2 can be applied here:

- the first proof, using diagonal constraints, applies as the PTA built above does not contain zero-delay cycles;
- the second proof 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 parametric clocks in that case.

B Proof of Theorem 14

Let us first recall below a useful result stating that the projection onto the parameters of a constraint can only become stricter along a run.

Lemma 28. *Let A be a PTA, and let ρ be a run of A reaching (l, C). Then, for any successor (l', C') of (l, C), we have $\mathsf{C}'{\downarrow_P} \subseteq \mathsf{C}{\downarrow_P}$.*

We recall below two results from [8].

Proposition 29. *Let A be a PTA, and let ρ be a run of A reaching (l, C). Let v be a parameter valuation. There exists an equivalent run in $v(\mathsf{A})$ iff $v \models \mathsf{C}{\downarrow_P}$.*

Proof. From [8, Propositions 3.17 and 3.18]. □

Proposition 30. *Let A be a PTA, let v be a parameter valuation. Let ρ be a run of $v(\mathsf{A})$ reaching (l, w).*

Then there exists an equivalent symbolic run in A reaching (l, C), with $v \models \mathsf{C}{\downarrow_P}$.

Proof. From [8, Proposition 3.18]. □

We will formally show the correctness of $\mathsf{TPSynth}$ in Theorem 14. Before that, we need some intermediate results.

Lemma 31. *Let A be a PTA, let v be a parameter valuation. Assume $\mathsf{TPSynth}(\mathsf{A}, v)$ 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 32. *Let* A *be a PTA, let* v *be a parameter valuation. Assume* $\mathsf{TPSynth}(\mathsf{A}, v)$ *terminates with constraint* K*. Then for all* $v' \models \mathsf{K}$*,* $\mathit{Traces}(v'(\mathsf{A})) = \mathit{Traces}(v(\mathsf{A}))$*.*

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

$\subseteq$ Let ρ' be a run of $v'(\mathsf{A})$, reaching a state (l, w'). From Proposition 30, there exists an equivalent run in A reaching a state (l, C'), with $v' \models \mathsf{C}'{\downarrow_P}$.
We will now prove by *reductio ad absurdum* that $v \models \mathsf{C}'{\downarrow_P}$. Assume $v \not\models \mathsf{C}'{\downarrow_P}$. Hence (l, C') is either a v-incompatible state met in $\mathsf{TPSynth}(\mathsf{A}, v)$, or the successor of some v-incompatible state met in $\mathsf{TPSynth}(\mathsf{A}, v)$.
1. Assume (l, C') is a v-incompatible state met in $\mathsf{TPSynth}(\mathsf{A}, v)$. By construction, $\mathsf{C}'{\downarrow_P}$ has been added to K_{bad} (line 1 in Algorithm 1), hence $\mathsf{C}'{\downarrow_P} \subseteq \mathsf{K}_{bad}$ hence $\neg\mathsf{K}_{bad} \cap \mathsf{C}'{\downarrow_P} = \emptyset$ hence $(\mathsf{K}_{good}\neg\mathsf{K}_{bad}) \cap \mathsf{C}'{\downarrow_P} = \emptyset$ hence $\mathsf{K} \cap \mathsf{C}'{\downarrow_P} = \emptyset$. This contradicts that $v' \models \mathsf{K}$.
2. Assume (l, C') is a v-incompatible state not met in $\mathsf{TPSynth}(\mathsf{A}, v)$, *i.e.* it belongs to some path starting from a v-incompatible state (l'', C'') met in $\mathsf{TPSynth}(\mathsf{A}, v)$. From Lemma 28, $\mathsf{C}'{\downarrow_P} \subseteq \mathsf{C}''{\downarrow_P}$, and hence $\mathsf{C}'{\downarrow_P} \subseteq \mathsf{C}''{\downarrow_P} \subseteq \mathsf{K}_{bad}$; then we apply the same reasoning as above to show that $\mathsf{K} \cap \mathsf{C}'{\downarrow_P} = \emptyset$, which contradicts that $v' \models \mathsf{K}$.

Hence $v \models \mathsf{C}'{\downarrow_P}$.
Now, from Proposition 29, there exists an equivalent run in $v(\mathsf{A})$, which gives that $\mathit{Traces}(v'(\mathsf{A})) \subseteq \mathit{Traces}(v(\mathsf{A}))$.

$\supseteq$ Let ρ be a run of $v(\mathsf{A})$, reaching a state (l, w). From Proposition 30, there exists an equivalent run in A reaching a state (l, C), with $v \models \mathsf{C}{\downarrow_P}$. From the fixpoint condition of Algorithm 1, all v-compatible states of A have been explored in $\mathsf{TPSynth}(\mathsf{A}, v)$, hence $(l, \mathsf{C}) \in S$, where S is the set of states explored just before termination of $\mathsf{TPSynth}(\mathsf{A}, v)$. By construction, $\mathsf{K} \subseteq \mathsf{C}{\downarrow_P}$; since $v' \models \mathsf{K}$ then $v' \models \mathsf{C}{\downarrow_P}$. Hence, from Proposition 29, there exists an equivalent run in $v'(\mathsf{A})$, which gives that $\mathit{Traces}(v'(\mathsf{A})) \supseteq \mathit{Traces}(v(\mathsf{A}))$. □

Theorem 14 (correctness of TPSynth). *Let* A *be a PTA, let* v *be a parameter valuation. Assume* $\mathsf{TPSynth}(\mathsf{A}, v)$ *terminates with constraint* K*. Then* $v \models \mathsf{K}$*, and for all* $v' \models \mathsf{K}$*,* $\mathit{Traces}(v'(\mathsf{A})) = \mathit{Traces}(v(\mathsf{A}))$*.*

Proof. From Lemmas 31 and 32. □

C Proof of Theorem 15

Theorem 15 (completeness of TPSynth). *Let* A *be a deterministic PTA, let* v *be a parameter valuation. Assume* $\mathsf{TPSynth}(\mathsf{A}, v)$ *terminates with constraint* K*. Then* $v' \models \mathsf{K}$ *iff* $\mathit{Traces}(v'(\mathsf{A})) = \mathit{Traces}(v(\mathsf{A}))$*.*

Proof.

$\Rightarrow$ From Theorem 14.

$\Leftarrow$ Let v' be a parameter valuation such that $\mathit{Traces}(v'(\mathsf{A})) = \mathit{Traces}(v(\mathsf{A}))$. 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 show a stronger result, that is $\mathsf{TPSynth}(\mathsf{A}, v) = \mathsf{TPSynth}(\mathsf{A}, v')$. Indeed, $\mathsf{TPSynth}(\mathsf{A}, v')$ proceeds by exploring states similarly to $\mathsf{TPSynth}(\mathsf{A}, v)$. From Proposition 29, the v-incompatible and v-compatible states met $\mathsf{TPSynth}(\mathsf{A}, v')$ will be the same as in $\mathsf{TPSynth}(\mathsf{A}, v)$, and hence the constraints K_{good} and K_{bad} will be the same too. Hence $\mathsf{TPSynth}(\mathsf{A}, v) = \mathsf{TPSynth}(\mathsf{A}, v')$, which trivially gives that $v' \models \mathsf{TPSynth}(\mathsf{A}, v)$. □

D Proofs of Section 5.1

D.1 Proof of Lemma 19

Lemma 19. *Let* A *be a 1cPTA. Let* (l, C) *be a reachable symbolic state of* A. *Then* $\mathsf{C} \in \mathsf{1CSC}(\mathsf{A})$.

Proof. By induction on the length of the runs.

Base case A run of length 0 consists of the sole initial state. According to the semantics of PTA, this state is $(l_{\mathsf{init}}, \mathsf{C}_{\mathsf{init}})$, where $\mathsf{C}_{\mathsf{init}}$ is $(X = 0)^{\uparrow} \wedge I(l_{\mathsf{init}})$, *i.e.* $x \geq 0 \wedge I(l_{\mathsf{init}})$. From Definition 17, $I(l_{\mathsf{init}})$ is of the form $\bigwedge_i lt_i \sim x$, with lt_i parametric linear terms of A, hence $I(l_{\mathsf{init}}) \in \mathsf{1CSC}(\mathsf{A})$. Furthermore, $x \geq 0$ obviously belongs to $\mathsf{1CSC}(\mathsf{A})$. Hence the initial constraint belongs to $\mathsf{1CSC}(\mathsf{A})$.

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

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

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

Guard From Definition 17, a guard is of the form $x \prec \sum_i \alpha_i p_i + c$, with $p_i \in P$ and $\alpha_i \in \mathbb{Z}$; hence $g \in \mathsf{1CSC}(\mathsf{A})$. Since $\mathsf{C} \in \mathsf{1CSC}(\mathsf{A})$ by induction hypothesis, then $\mathsf{C} \wedge g \in \mathsf{1CSC}(\mathsf{A})$.

Reset Then, $[(\mathsf{C} \wedge g)]_R$ is equivalent to removing x in $\mathsf{C} \wedge g$ (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.* $\bigwedge_j lt_j^1 \sim lt_j^2$) unchanged. As for the inequalities containing x (*i.e.* $\bigwedge_i lt_i \sim x$), 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 $lt_i \sim lt_{i'}$, which will be added to the set of parametric inequalities. Finally, adding $x = 0$ (which belongs to $\mathsf{1CSC}(\mathsf{A})$) makes $[(\mathsf{C} \wedge g)]_R$ remain in $\mathsf{1CSC}(\mathsf{A})$.

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 $\big([(\mathsf{C} \wedge g)]_R\big)^{\uparrow}$ still belong to $\mathsf{1CSC}(\mathsf{A})$.

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

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

D.2 Proof of Proposition 21

Proposition 21. *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 $x \prec lt_i$ cannot be conjuncted with other $x \prec' lt_i$, where $\prec \neq \prec'$ (unless $\prec = \geq$ and $\prec' = \leq$ 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 $x \prec lt_i$. The same reasoning applies to parametric inequalities $lt_j^1 \sim lt_j^2$.

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

Hence the set $\mathsf{1CSC}(\mathsf{A})$ 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 A contains at most $|L| \times 2^{|LT|(|LT|+1)}$ symbolic states. □

E Proof of Theorem 24

The PTA gadget depicted in Fig. 4 can be characterized in the following lemma.

Lemma 33. *In the PTA gadget depicted in Figure 4,* l_2 *is reachable and* b *can never fire iff* $p_l = p_u$.

Proof.

⇒ Assume l_2 is reachable; hence, from the guards and invariants, we necessarily have $p_l \leq p_u$. Furthermore, b can fire iff it is possible to stay a non-null duration in l_1 iff $p_l < p_u$. Hence, b cannot fire implies $p_l \geq p_u$.

⇐ Assume $p_l = p_u$. Then no time can elapse in l_1, and hence b cannot fire. Furthermore, l_2 is obviously reachable for any such parameter valuation. □

Theorem 24. *The language-preservation problem is undecidable for L/U-PTA with at least one lower-bound and at least one upper-bound parameter.*

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

First, let us rewrite the 2CM encoding of Section 3.1 using L/U-PTA as follows. We split the parameter p used in the PTA A in the proof of Theorem 6 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 4 before the initial state of our modified encoding of the proof of Theorem 6; more precisely, we fuse l_2 in Fig. 4 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 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 4 is aa. Recall that in the proof of Theorem 6, the language of $p = 0$ is a^ω, and hence the language of our modified PTA A_{LU} for v is $aaa^\omega = a^\omega$.

Suppose the 2CM does not halt, and consider a parameter valuation $v' \neq v$. If $p_l \neq p_u$ in v', then from Lemma 33, 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 $v'(\mathsf{A}_{LU})$ differs from the language of $v(\mathsf{A}_{LU})$ (that is a^ω). If $p_l = p_u$, then we fall in the situation of Theorem 6: 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 2CM 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 2CM halts iff there exists a parameter valuation $v' \neq v$ such that the language is the same as for v. □