Parametric model checking timed automata under non-Zenoness assumption[*]

Étienne André[1], Hoang Gia Nguyen[1], Laure Petrucci[1], and Jun Sun[2]

[1] LIPN, CNRS UMR 7030, Université Paris 13
99, avenue Jean-Baptiste Clément, F-93430 Villetaneuse, France
[2] ISTD, Singapore University of Technology and Design, Singapore

Abstract. Real-time systems often involve hard timing constraints and concurrency, and are notoriously hard to design or verify. Given a model of a real-time system and a property, parametric model-checking aims at synthesizing timing valuations such that the model satisfies the property. However, the counter-example returned by such a procedure may be *Zeno* (an infinite number of discrete actions occurring in a finite time), which is unrealistic. We show here that synthesizing parameter valuations such that at least one counterexample run is non-Zeno is undecidable for parametric timed automata (PTAs). Still, we propose a semi-algorithm based on a transformation of PTAs into *Clock Upper Bound PTAs* to derive all valuations whenever it terminates, and some of them otherwise.

Keywords: parameter synthesis, Zeno behaviors, parametric timed automata

1 Introduction

Timed automata (TAs) [AD94] are a popular formalism for real-time systems modeling and verification, providing explicit manipulation of clock variables. Real-time behavior is captured by clock constraints on system transitions, setting or resetting clocks, etc. TAs have been studied in various settings (such as planning [KMH01]) and benefit from powerful tools such as Uppaal [LPY97] or PAT [SLDP09]. Verification tools for TA based models have proven to be successful [LPY97,BDM+98,Wan02,BLN03].

Model checking TAs consists of checking whether there exists an accepting cycle (i. e. a cycle that visits infinitely often a given set of locations) in the automaton made of the product of the TA modeling the system with the TA representing a violation of the desired property (often the negation of a property expressed, e. g. in CTL). However, such an accepting cycle does not necessarily

[*] This is the author (and partially extended) version of the manuscript of the same name published in the proceedings of the 9th NASA Formal Methods Symposium (NFM 2017). The final version is available at dx.doi.org/10.1007/978-3-319-57288-8_3. This work is partially supported by the ANR national research program PACS (ANR-14-CE28-0002).

mean that the property is violated: indeed, a known problem of TAs is that they allow Zeno behaviors. An infinite run is non-Zeno if it takes an unbounded amount of time; otherwise it is Zeno. Zeno runs are infeasible in reality and thus must be pruned during system verification. That is, it is necessary to check whether a run is Zeno or not so as to avoid presenting Zeno runs as counterexamples. The problem of checking whether a timed automaton accepts at least one non-Zeno run, i. e. the emptiness checking problem, has been tackled previously (e. g. [Tri99,TYB05,BG06,GB07,HSW12,WSW+15]).

It is often desirable not to fix *a priori* all timing constants in a TA: either for tuning purposes, or to evaluate robustness when clock values are imprecise. For that purpose, parametric timed automata (PTAs) extend TAs with parameters [AHV93]. Although most problems of interest are undecidable for PTAs [And15], some (semi-)algorithms were proposed to tackle practical parameter synthesis (e. g. [ACEF09,KP12,JLR15,ABB+16]). We address here the synthesis of parameter valuations for which there exists a non-Zeno cycle in a PTA; this is highly desirable when performing parametric model-checking for which the parameter valuations violating the property should not allow only Zeno-runs. As far as the authors know, this is the first work on parametric model checking of timed automata with the non-Zenoness assumption. Just as for TAs, the parametric zone graph of PTAs (used in e. g. [HRSV02,ACEF09,JLR15]) cannot be used to check whether a cycle is non-Zeno. Therefore, we propose here a technique based on *clock upper bound PTAs* (CUB-PTAs), a subclass of PTAs satisfying some syntactic restriction, and originating in CUB-TAs for which the non-Zeno checking problem is most efficient [WSW+15]. In contrast to regular PTAs, we show that synthesizing valuations for CUB-PTAs such that there exists an infinite non-Zeno cycle can be done based on (a light extension of) the parametric zone graph.

We make the following technical contributions in this work:

1. We show that the parameter synthesis problem for PTAs with non-Zenoness assumption is undecidable.
2. We show that any PTA can be transformed into a finite list of CUB-PTAs;
3. We develop a semi-algorithm to solve the non-Zeno synthesis problem using CUB-PTAs, implemented in IMITATOR and validated using benchmarks.

Outline The paper is organized as follows. Section 2 recalls the necessary preliminaries, i. e. the notations, definitions of PTAs, and explicits the two problems we address: non-Zeno-Büchi emptiness and non-Zeno parameter synthesis. Then Section 3 shows the undecidability of non-Zeno-Büchi emptiness. We then present the concept of CUB-PTAs (Section 4), and show how to transform a PTA into a list of CUB-PTAs. Zeno-free parametric model-checking of CUB-PTA is addressed in Section 5, and experiments reported in Section 6. Finally, Section 7 concludes and gives perspectives for future work.

2 Preliminaries

This section recalls the different concepts we base on (clocks, parameters, constraints and Parametric Timed Automata) before introducing the problems we address.

2.1 Clocks, parameters and constraints

Throughout this paper, we assume a set $X = \{x_1, \dots, x_H\}$ of *clocks*, i. e. real-valued variables that evolve at the same rate. A clock valuation is a function $w : X \to \mathbb{R}_{\geq 0}$. We identify a clock valuation w with the *point* $(w(x_1), \dots, w(x_H))$. We write $X = 0$ for $\bigwedge_{1 \leq i \leq H} x_i = 0$. Given $d \in \mathbb{R}_{\geq 0}$, $w + d$ denotes the valuation such that $(w + d)(x) = w(x) + d$, for all $x \in X$.

We assume a set $P = \{p_1, \dots, p_M\}$ of *parameters*, i. e. unknown constants. A parameter *valuation* v is a function $v : P \to \mathbb{Q}_{\geq 0}$. We identify a valuation v with the *point* $(v(p_1), \dots, v(p_M))$. A *strictly positive* parameter valuation is a valuation $v : P \to \mathbb{Q}_{>0}$.

In the following, we assume $\triangleleft \in \{<, \leq\}$ and $\bowtie \in \{<, \leq, \geq, >\}$. Throughout this paper, *lt* denotes a linear term over $X \cup P$ of the form $\sum_{1 \leq i \leq H} \alpha_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + d$, with $\alpha_i, \beta_j, d \in \mathbb{N}$. Similarly, *plt* denotes a parametric linear term over P, that is a linear term without clocks ($\alpha_i = 0$ for all i). A *constraint* C (i. e. a convex polyhedron) over $X \cup P$ is a set of inequalities of the form $lt \bowtie lt'$, with lt, lt' two linear terms. We denote by *true* (resp. *false*) the constraint that corresponds to the set of all possible (resp. the empty set of) valuations. Given a parameter valuation v, $v(C)$ denotes the constraint over X obtained by replacing each parameter p in C with $v(p)$. Likewise, given a clock valuation w, $w(v(C))$ denotes the expression obtained by replacing each clock x in $v(C)$ with $w(x)$. We say that v *satisfies* C, denoted by $v \models C$, if the set of clock valuations satisfying $v(C)$ is non-empty. We say that C is *satisfiable* if $\exists w, v$ s.t. $w(v(C))$ evaluates to true. We define the *time elapsing* of C, denoted by $C^{\nearrow}$, as the constraint over X and P obtained from C by delaying all clocks by an arbitrary amount of time. Given $R \subseteq X$, we define the *reset* of C, denoted by $[C]_R$, as the constraint obtained from C by resetting the clocks in R, and keeping the other clocks unchanged. We denote by $C{\downarrow_P}$ the projection of C onto P, i. e. obtained by eliminating the clock variables using existential quantification.

A *guard* g is a constraint over $X \cup P$ defined by inequalities of the form $x \bowtie plt$. We assume w.l.o.g. that, in each guard, given a clock x, at most one inequality is in the form $x \triangleleft plt$, that is a clock has a single upper bound (or none). A non-parametric guard is a guard over X, i. e. with inequalities $x \bowtie z$, with $z \in \mathbb{N}$. A *parametric zone* C is a constraint over $X \cup P$ defined by inequalities of the form $x_i - x_j \bowtie plt$. A *parametric constraint* K is a constraint over P defined by inequalities of the form $plt \bowtie plt'$, with plt, plt' two parametric linear terms. We use the notation $v \models K$ to indicate that valuating parameters p with $v(p)$ in K evaluates to true. We denote by $\top$ (resp. $\bot$) the parametric constraint that corresponds to the set of all possible (resp. the empty set of) parameter valuations. Given two parametric constraints K_1 and K_2, we write $K_1 \subseteq K_2$ whenever for all v, $v \models K_1 \Rightarrow v \models K_2$.

2.2 Parametric timed automata

Parametric timed automata (PTA) extend timed automata with parameters within guards in place of integer constants [AHV93].

Syntax

Definition 1. *A PTA $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, l_0, X, P, K_0, I, E)$, where:*

1. *Σ is a finite set of actions,*
2. *L is a finite set of locations,*
3. *$l_0 \in L$ is the initial location,*
4. *X is a set of clocks,*
5. *P is a set of parameters,*
6. *K_0 is the initial parameter constraint,*
7. *I is the invariant, assigning to every $l \in L$ a guard $I(l)$,*
8. *E is a set of edges $e = (l, g, a, R, l')$ where $l, l' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq X$ is a set of clocks to be reset, and g is a guard.*

The initial constraint K_0 is used to constrain some parameters (as in, e.g. [HRSV02,ACEF09]); in other words, it defines a domain of valuation for the parameters. For example, given two parameters $p_{\min}$ and $p_{\max}$, we may want to ensure that $p_{\min} \leq p_{\max}$. Given $\mathcal{A} = (\Sigma, L, l_0, X, P, K_0, I, E)$, we write $\mathcal{A}.K_0$ as a shortcut for the initial constraint of $\mathcal{A}$. In addition, given K'_0, we denote by $\mathcal{A}(K'_0)$ the PTA where $\mathcal{A}.K_0$ is replaced with K'_0.

Observe that, as in [WSW$^+$15], we do not define accepting locations. In our work, we are simply interested in computing valuations for which there is a non-Zeno cycle. A more realistic parametric model checking approach would require additionally that the cycle is accepting, i.e. it contains at least one accepting location. However, this has no specific theoretical interest, and would impact the readability of our exposé.

Given a parameter valuation $v \models \mathcal{A}.K_0$, we denote by $v(\mathcal{A})$ the non-parametric TA where all occurrences of a parameter p_i have been replaced by $v(p_i)$. If $v \not\models \mathcal{A}.K_0$, we assume the model is not defined (i.e. it corresponds to an empty TA, with no location).

Definition 2 (Concrete semantics of a TA). *Given a PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, K_0, I, E)$, and a parameter valuation v, the concrete semantics of $v(\mathcal{A})$ is given by the timed transition system $(S, s_0, \rightarrow)$, with $S = \{(l, w) \in L \times \mathbb{R}^H_{\geq 0} \mid w(v(I(l)))$ is true$\}$, $s_0 = (l_0, \mathbf{0})$, and $\rightarrow$ consists of the discrete and (continuous) delay transition relations:*

- *discrete transitions: $(l, w) \overset{e}{\rightarrow} (l', w')$, if $(l, w), (l', w') \in S$, there exists $e = (l, g, a, R, l') \in E$, $w' = [w]_R$, and $w(v(g))$ is true.*

– *delay transitions:* $(l, w) \xrightarrow{d} (l, w+d)$, *with* $d \in \mathbb{R}_{\geq 0}$, *if* $\forall d' \in [0, d], (l, w+d') \in S$.

A *(concrete) run* is a sequence $r = s_0 \alpha_0 s_1 \alpha_1 \cdots s_n \alpha_n \cdots$ s.t. $\forall i, (s_i, \alpha_i, s_{i+1}) \in \rightarrow$. We consider as usual that concrete runs strictly alternate delays d_i and discrete transitions e_i and we thus write concrete runs in the form $r = s_0 \overset{(d_0,e_0)}{\rightarrow} s_1 \overset{(d_1,e_1)}{\rightarrow} \cdots$. We refer to a state of a run starting from the initial state of a TA $\mathcal{A}$ as a *concrete state* of $\mathcal{A}$. Note that when a run is finite, it must end with a concrete state. Given a concrete state $s = (l, w)$, we say that s is reachable (or that $v(\mathcal{A})$ reaches s) if s belongs to a run of $v(\mathcal{A})$. By extension, we say that l is reachable in $v(\mathcal{A})$, if there exists a concrete state (l, w) that is reachable.

An infinite run is said to be *Zeno* if it contains an infinite number of discrete transitions within a finite delay, i. e. if the sum of all delays d_i is bounded.

Symbolic semantics Let us recall the symbolic semantics of PTAs (as in e. g. [ACEF09,JLR15]). A symbolic state is a pair $\mathbf{s} = (l, C)$ where $l \in L$ is a location, and C its associated parametric zone.

The initial symbolic state of $\mathcal{A}$ is $\mathbf{s}_0 = \big(l_0, (\{\mathbf{0}\} \wedge I(l_0))^{\nearrow} \wedge I(l_0) \wedge K_0\big)$. That is, the initial state corresponds to all clocks equal to 0 followed by time-elapsing, intersected with the initial invariant and the initial parameter constraint.

The symbolic semantics relies on the Succ operation. Given a symbolic state $\mathbf{s} = (l, C)$ and an edge $e = (l, g, a, R, l')$, $\mathsf{Succ}(\mathbf{s}, e) = (l', C')$, with $C' = \big([(C \wedge g)]_R\big)^{\nearrow}$. The Succ operation is effectively computable, using polyhedra operations: note that the successor of a parametric zone C is a parametric zone.

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

The symbolic semantics is often given in the form of a *parametric zone graph*, i. e. symbolic states of $\mathcal{A}$ and transitions $(\mathbf{s}, e, \mathbf{s}')$ whenever $\mathbf{s}' = \mathsf{Succ}(\mathbf{s}, e)$. Given a symbolic run $(l_0, C_0) \overset{e_0}{\Rightarrow} (l_1, C_1) \overset{e_1}{\Rightarrow} \cdots \overset{e_{n-1}}{\Rightarrow} (l_n, C_n) \cdots$, its *untimed support* is the sequence $l_0 e_0 l_1 \cdots e_{n-1} l_n \cdots$. Two runs (symbolic or concrete) are *equivalent* if they have the same untimed support.

Let us recall a lemma relating concrete and symbolic runs.

Lemma 1. *Let $\mathcal{A}$ be a PTA, and let $\mathbf{r}$ be a symbolic run of $\mathcal{A}$ reaching (l, C). Let $v \models \mathcal{A}.K_0$ be a parameter valuation. There exists an equivalent concrete run in $v(\mathcal{A})$ iff $v \models C{\downarrow_P}$.*

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

Given a symbolic run $\mathbf{r}$ reaching (l, C), we call the *concrete runs associated with* $\mathbf{r}$ the concrete runs equivalent to $\mathbf{r}$ in $v(\mathcal{A})$, for all $v \models C{\downarrow_P}$.

2.3 Problems

In this paper, we aim at addressing the following two problems. They both concern the existence of an infinite non-Zeno run. The first one aims at checking whether the set of parameter valuations leading to such a run is empty, while the second synthesizes such a set of valuations.

non-Zeno emptiness problem:
INPUT: A PTA $\mathcal{A}$
PROBLEM: Is the set of parameter valuations v for which there exists a non-Zeno infinite run in $v(\mathcal{A})$ empty?

non-Zeno synthesis problem:
INPUT: A PTA $\mathcal{A}$
PROBLEM: Synthesize the set of parameter valuations v for which there exists an infinite non-Zeno run in $v(\mathcal{A})$.

3 Undecidability of the non-Zeno emptiness problem

As reachability is undecidable for PTAs [AHV93], it is unsurprising that the existence of at least a parameter valuation for which there exists a non-Zeno infinite run is undecidable too. The result holds with as few as one parameter, even bounded (typically in $[0, 1]$). Let us formalize this result below.

Theorem 1. *The non-Zeno emptiness problem is undecidable for PTAs with at least four clocks and one (bounded) parameter.*

Proof. By reduction from the halting problem of a deterministic 2-counter-machine, which is undecidable [Min67]. We encode a 2-counter machine (2CM) using PTAs, following an encoding in [AM15]. This encoding is such that the location $l_{\texttt{halt}}$ encoding the halting state of the 2CM is reachable iff the 2CM halts, and for valuations of the (unique) parameter v such that $v(p)$ is larger than or equal to the maximum value of the counters along the (unique) run of the machine. Then, since this encoding is such that for any parameter valuation, the encoding stops after $v(p)$ discrete steps, the encoding has no infinite run for any valuation.

Then, from the location encoding the halting location (i. e. $l_{\texttt{halt}}$), we add a transition resetting x to a new location l_f. This location has a self-loop guarded with $x = 1$ and resetting x (where x is any of the four clocks used in the encoding in [AM15]). Hence whenever $l_{\texttt{halt}}$ is reachable, there is an infinite non-Zeno run looping on l_f. That is, there is an infinite non-Zeno run iff the 2CM halts.

A full proof is available in Appendix A. ■

Since the emptiness problem is undecidable, the synthesis problem becomes intractable. In the remainder of this paper, we will devise a *semi-algorithm* to address the non-Zeno synthesis problem, i. e. an algorithm that computes the exact solution if it terminates. Otherwise, we will compute an under-approximation of the result.

4 CUB-parametric timed automata

4.1 CUB timed automata

It has been shown (e. g. [BG06,Tri99]) that checking whether a run of TA is infeasible based on the symbolic semantics alone. In [WSW+15], the authors identified a subclass of TAs called CUB-TAs for which non-Zenoness checking based on the symbolic semantics is feasible. Furthermore, they show that not only CUB-TAs are expressive enough to model most of the benchmark timed systems, but more importantly, an arbitrary TA can be transformed into a CUB-TA. Based on their work, we first show that arbitrary PTAs can be transformed into a parametric version of CUB-TAs, and then solve the non-Zeno synthesis problem based on parametric CUB-TAs.

As defined in [WSW+15], a clock upper bound is either ∞ or a pair $(n, \triangleleft)$ where $n \in \mathbb{Q}$ (recall that $\triangleleft$ is either $<$ or $\leq$). We write $(n_1, \triangleleft_1) = (n_2, \triangleleft_2)$ to denote $n_1 = n_2$ and $\triangleleft_1 = \triangleleft_2$; $(n_1, \triangleleft_1) \leq (n_2, \triangleleft_2)$ to denote $n_1 < n_2$, or if $n_1 = n_2$, then either $\triangleleft_2$ is $\leq$ or both $\triangleleft_1$ and $\triangleleft_2$ are $<$. Further, we write $(n, \triangleleft) > d$ where d is a constant to denote $n > d$. We define $\min((n, \triangleleft_1), (m, \triangleleft_2))$ to be $(n, \triangleleft_1)$ if $(n, \triangleleft_1) \leq (m, \triangleleft_2)$, and $(m, \triangleleft_2)$ otherwise. Given a clock x and a non-parametric guard g, we write $ub(g, x)$ to denote the upper bound of x given g. Formally,

$$ub(g,x) = \begin{cases} (n, \triangleleft) & \text{if } g \text{ is } x \triangleleft n \\ \infty & \text{if } g \text{ is } x > n \text{ or } x \geq n \\ \infty & \text{if } g \text{ is } x' \bowtie n \text{ and } x' \neq x \\ \infty & \text{if } g \text{ is } \mathit{true} \\ \min(ub(g_1, x), ub(g_2, x)) & \text{if } g \text{ is } g_1 \wedge g_2 \end{cases}$$

Let us now introduce CUB-TAs.[3]

Definition 3. *A TA is a* CUB-TA *if for each edge (l, g, a, R, l'), for all clocks $x \in X$, we have*

1. *$ub(I(l), x) \leq ub(g, x)$, and*
2. *if $x \notin R$, then $ub(I(l), x) \leq ub(I(l'), x)$.*

Intuitively, every clock in a CUB-TA has a non-decreasing upper bound along any path until it is reset.

4.2 Parametric clock upper bounds

Let us define clock upper bounds in a parametric setting. A *parametric clock upper bound* is either ∞ or a pair $(plt, \triangleleft)$.

Given a clock x and a guard g, we denote by $pub(g, x)$ the parametric upper bound of x given g. This upper bound is a parametric linear term. Formally,

[3] Note that our definition is slightly more liberal than that in [WSW+15].

$$pub(g,x) = \begin{cases} (plt, \lhd) & \text{if } g \text{ is } x \lhd plt \\ \infty & \text{if } g \text{ is } x > plt \text{ or } x \geq plt \\ \infty & \text{if } g \text{ is } x' \bowtie plt \text{ and } x' \neq x \\ \infty & \text{if } g \text{ is } \mathit{true} \\ \min(pub(g_1,x), pub(g_2,x)) & \text{if } g \text{ is } g_1 \wedge g_2 \end{cases}$$

Recall that, in each guard, given a clock x, at most one inequality is in the form $x \lhd plt$. In that case, at most one of the two terms is not ∞ and therefore the minimum (last case) is well-defined (with the usual definition that $\min(plt, \infty) = plt$).[4]

We write $(plt_1, \lhd_1) \leq (plt_2, \lhd_2)$ to denote the constraint

$$\begin{cases} plt_1 < plt_2 \text{ if } \lhd_1 = \leq \text{ and } \lhd_2 = < \\ plt_1 \leq plt_2 \text{ otherwise.} \end{cases}$$

That is, we constrain the first parametric clock upper bound to be smaller than or equal to the second one, depending on the comparison operator.

Given two parametric clock upper bounds $pcub_1$ and $pcub_2$, we write $pcub_1 \leq pcub_2$ to denote the constraint

$$\begin{cases} (plt_1, \lhd_1) \leq (plt_2, \lhd_2) & \text{if } pcub_1 = (plt_1, \lhd_1) \text{ and } pcub_2 = (plt_2, \lhd_2) \\ \top & \text{if } pcub_2 = \infty \\ \bot & \text{otherwise.} \end{cases}$$

This yields an inequality constraining the first parametric clock upper bound to be smaller than or equal to the second one.

4.3 CUB parametric timed automata

We extend the definition of CUB-TAs to parameters as follows:

Definition 4. *A PTA is a* CUB-PTA *if for each edge (l, g, a, R, l'), for all clocks $x \in X$, the following conditions hold:*

1. $\mathcal{A}.K_0 \subseteq \big(pub(I(l), x) \leq pub(g, x)\big)$, *and*
2. *if $x \notin R$, then* $\mathcal{A}.K_0 \subseteq \big(pub(I(l), x) \leq pub(I(l'), x)\big)$.

Hence, a PTA is a *CUB-PTA* iff every clock has a non-decreasing upper bound along any path before it is reset, for all parameter valuations satisfying the initial constraint $\mathcal{A}.K_0$.

Note that, interestingly enough, the class of hardware circuits modeled using a bi-bounded inertial delay[5] fits into CUB-PTAs (for all parameter valuations).

[4] Note that if a clock has more than a single upper bound in a guard, then the minimum can be encoded as a disjunction of constraints, and our results would still apply with non-convex constraints (that can be implemented using a finite list of convex constraints).

[5] This model assumes that, after the change of a signal in the input of a gate, the output changes after a delay which is modeled using a parametric closed interval.

Example 1. Consider the PTA $\mathcal{A}$ in Fig. 1a s.t. $\mathcal{A}.K_0 = \top$. Then $\mathcal{A}$ is not CUB: for x, the upper bound in l_0 is $x \leq 1$ whereas that of the guard on the transition outgoing l_0 is $x \leq p$. $(1, \leq) \leq (p, \leq)$ yields $1 \leq p$. Then, $\top \not\subseteq \big(1 \leq p\big)$; for example, $p = 0$ does not satisfy $1 \leq p$. □

Example 2. Consider again the PTA $\mathcal{A}$ in Fig. 1a, this time assuming that $\mathcal{A}.K_0 = (p = 1 \land 1 \leq p' \land p' \leq p'')$. This PTA is a CUB-PTA. (The largest constraint K_0 making this PTA a CUB will be computed in Example 4.) □

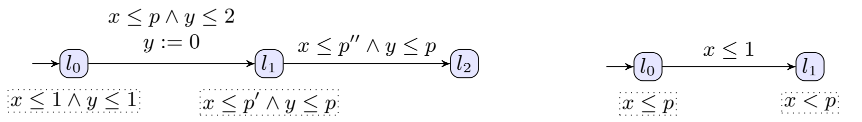
Example 3. The three examples of **??**, introduce all possible cases encountered in Algorithm 1. Fig. 2a features a self-loop on the initial location l_0. The CUB-PTA conditions enforce a constraint $p_1 \leq p_2$. Therefore, the model is CUB-PTA for all valuations of p_1 and p_2 such that $p_1 \leq p_2$.

Contrarily, in Fig. 2b, the edge from the initial location l_0 to location l_1 (or location l_2) induces a constraint $\infty \leq p_2$ (respectively $\infty < p_1$, which is always false. Therefore, the model is not a CUB-PTA, whatever the valuation.

Finally, the example of Fig. 2c is CUB-PTA for all valuations of p. □

Lemma 2. *Let $\mathcal{A}$ be a CUB-PTA. Let $v \models \mathcal{A}.K_0$ be a parameter valuation. Then $v(\mathcal{A})$ is a CUB-TA.*

Proof. Let $v \models \mathcal{A}.K_0$. Let $e = (l, g, a, R, l')$ be an edge. Given a clock $x \in X$, from Definition 4, we have that $v \models \big(pub(I(l), x) \leq pub(g, x)\big)$, and therefore $v(pub(I(l), x)) \leq v(pub(g, x))$. This matches the first case of Definition 3. The second case ($x \notin R$) is similar. ■


$x \leq p \land y \leq 2$
$y := 0$
$x \leq p'' \land y \leq p$
l_0
l_1
l_2
$x \leq 1 \land y \leq 1$
$x \leq p' \land y \leq p$
$x \leq 1$
l_0
l_1
$x \leq p$
$x < p$

(a) CUB for some valuations (b) CUB for no valuations

Fig. 1: Examples of PTAs to illustrate the CUB concept

4.4 CUB PTA detection

Given an arbitrary PTA, our approach works as follows. Firstly, we check whether it is a CUB-PTA for some valuations. If it is, we proceed to the synthesis problem, using the cycle detection synthesis algorithm (Section 5); however, the result may be partial, as it will only be valid for the valuations for which the PTA is CUB. This incompleteness may come at the benefit of a more efficient synthesis. If it is CUB for no valuation, it has to be transformed into an equivalent CUB-PTA (which will be considered in Section 4.5).

Our procedure to detect whether a PTA is CUB for some valuations is given in Algorithm 1. For each edge in the PTA, we enforce the CUB condition on each

Algorithm 1: $\mathsf{CUBdetect}(\mathcal{A})$

```
    Input: PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, K_0, I, E)$
    Output: A constraint $K$ ensuring the PTA is a CUB-PTA
1   $K \leftarrow K_0$
2   foreach edge $(l, g, a, R, l')$ do
3       foreach clock $x \in X$ do
4           $K \leftarrow K \wedge \big(pub(I(l), x) \leq pub(g, x)\big)$
5           if $x \notin R$ then $K \leftarrow K \wedge \big(pub(I(l), x) \leq pub(I(l'), x)\big)$
6   return $K$
```

clock by constraining the upper bound in the invariant of the source location to be smaller than or equal to the upper bound of the edge guard (line 4). Additionally, if the clock is not reset along this edge, then the upper bound of the source location invariant should be smaller than or equal to that of the target location (line 5). If the resulting set of constraints accepts parameter valuations (i. e. is not empty), then the PTA is a CUB-PTA for these valuations.

Example 4. Consider again the PTA $\mathcal{A}$ in Fig. 1a, assuming that $\mathcal{A}.K_0 = \top$. This PTA is CUB for $1 \leq p \wedge 1 \leq p' \wedge p' \leq p''$. □

Example 5. Consider the PTA $\mathcal{A}$ in Fig. 1b, with $\mathcal{A}.K_0 = \top$. When handling location l_0 and clock x, line 4 yields $\mathcal{A}.K = \top \wedge [(p, \leq) \leq (1, \leq)] = p \leq 1$ and then, from line 5, $\mathcal{A}.K = p \leq 1 \wedge [(p, \leq) \leq (p, <)] = p \leq 1 \wedge p < p = \bot$. Hence, there is no valuation for which this PTA is CUB. □

Proposition 1. *Let $\mathcal{A}$ be a PTA.*
Let $K = \mathsf{CUBdetect}(\mathcal{A})$. Then $\mathcal{A}(K)$ is a CUB-PTA.

Proof. From the fact that Algorithm 1 gathers constraints to match Definition 4. ■

4.5 Transforming a PTA into a disjunctive CUB-PTA

In this section, we show that an arbitrary PTA can be transformed into an extension of CUB-PTAs (namely *disjunctive CUB-PTA*), while preserving the symbolic runs.

For non-parametric TAs, it is shown in [WSW+15] that any TA can be transformed into an equivalent CUB-TA. This does not lift to CUB-PTAs.

Example 6. No equivalent CUB-PTA exists for the PTA in Fig. 2b where $K_0 = \top$. Indeed, the edge from l_1 to l_2 (resp. l_3) requires $p_1 \leq p_2$ (resp. $p_1 > p_2$). It is impossible to transform this PTA into a PTA where K_0 (which is $\top$) is included in both $p_1 \leq p_2$ and $p_1 > p_2$. □

Therefore, in order to overcome this limitation, we propose an alternative definition of *disjunctive CUB-PTAs*. They can be seen as a union (as defined in the timed automata patterns of, e. g. [DHQ+08]) of CUB-PTAs.

Definition 5. *A* disjunctive CUB-PTA *is a list of CUB-PTAs.*

Given a disjunctive CUB-PTA $\mathcal{A}_1, \ldots, \mathcal{A}_n$*, with* $\mathcal{A}_i = (\Sigma_i, L_i, l_0^i, X_i, P_i, K_0^i, I_i, E_i)$*, the PTA* associated with *this disjunctive PTA is*

$\mathcal{A} = (\bigcup_i \Sigma_i, \bigcup_i L_i \cup \{l_0\}, l_0, \bigcup_i X_i, \bigcup_i P_i, \bigcup_i K_0^i, \bigcup_i I_i, E)$*, where* $E = \bigcup_i E_i \cup E'$ *with* $E' = \bigcup_i (l_0, K_0^i, \epsilon, X, l_0^i)$*.*

Basically, the PTA associated with a disjunctive CUB-PTA is just an additional initial location that connects to each of the CUB-PTAs initial locations, with its initial constraint on the guard.[6]

Example 7. In Fig. 2d (without the dotted, blue elements), two CUB-PTAs are depicted, one (say $\mathcal{A}_1$) on the left with locations superscripted by 1, and one (say $\mathcal{A}_2$) on the right superscripted with 2. Assume $\mathcal{A}_1.K_0$ is $p_1 \leq p_2$ and $\mathcal{A}_2.K_0$ is $p_1 > p_2$. Then the full Fig. 2d (including dotted elements) is the PTA associated with the disjunctive CUB-PTA made of $\mathcal{A}_1$ and $\mathcal{A}_2$. $\square$

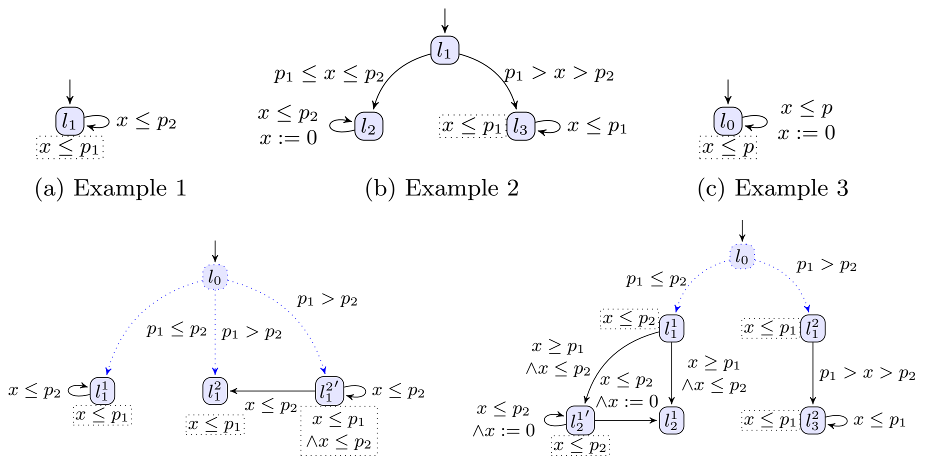
The key idea behind the transformation from a TA into a CUB-TA in [WSW+15] is as follows: whenever a location l is followed by an edge e and a location l' for which $ub(g, x) < ub(l, x)$ or $ub(l', x) < ub(l, x)$ for some x if $x \notin R$, otherwise $ub(g, x) < ub(l, x)$, location l is split into two locations: one (say l_1) with a "decreased upper bound", i. e. $x \lhd ub(l', x)$, that is then connected to l'; and one (say l_2) with the same invariant as in l, and with no transition to l'. Therefore, the original behavior is maintained. Note that this transformation induces some non-determinism (one must non-deterministically choose whether one enters l_1 or l_2, which will impact the future ability to enter l') but this has no impact on the existence of a non-Zeno cycle.

Here, we extend this principle to CUB-PTAs. A major difference is that, in the parametric setting, comparing two clock upper bounds does not give a Boolean answer but a parametric answer. For example, in a TA, $(2, \leq) \leq (3, <)$ holds (this is true), whereas in a PTA $(p_1, \leq) \leq (p_2, <)$ denotes the *constraint* $p_1 < p_2$. Therefore, the principle of our transformation is that, whenever we have to compare two parametric clock upper bounds, we consider both cases: here either $p_1 < p_2$ (in which case the first location does not need to be split) or $p_1 \geq p_2$ (in which case the first location shall be split). This yields a finite list of CUB-PTAs: each of these CUB-PTAs consists in one particular ordering of all parametric linear terms used as upper bounds in guards and invariants. (In practice, in order to reduce the complexity, we only define an order on the parametric linear terms the comparison of which is needed during the transformation process.)

The full transformation algorithm $\mathsf{CUBtrans}(\mathcal{A})$ is given in Appendix C.

Example 8. Let us transform the PTA in Fig. 2a: if $p_1 \leq p_2$ then the PTA is already CUB, and l_1 does not need to be split. This yields a first CUB-PTA, depicted on the left-hand side of Fig. 2d. However, if $p_1 > p_2$, then l_1 needs to

[6] A purely parametric constraint (e. g. $p_1 > p_2 \land p_3 = 3$) is generally not allowed by the PTA syntax, but can be simulated using appropriate clocks (e. g. $p_1 > x > p_2 \land p_3 = x' = 3$). Such parametric constraints are allowed in the input syntax of IMITATOR.



(a) Example 1 (b) Example 2 (c) Example 3

(d) Transformed version of Fig. 2a (e) Transformed version of Fig. 2b

Fig. 2: Examples: detection of and transformation into CUB-PTAs

be split into $l_1^{2\prime}$ (where time cannot go beyond p_2) and into l_1^2 (where time can go beyond p_2, until p_1), but the self-loop cannot be taken anymore (otherwise the associated guard makes the PTA not CUB). This yields a second CUB-PTA, depicted on the right-hand side of Fig. 2d. Both make a disjunctive CUB-PTA equivalent to Fig. 2a.

Similarly, we give the transformation of Fig. 2b in Fig. 2e. □

5 Zeno-free cycle synthesis in CUB-PTAs

Taking a disjunctive CUB-PTA as input, we show in this section that synthesizing the parameter valuations for which there exists at least one non-Zeno cycle (and therefore an infinite non-Zeno run) reduces to an SCC (strongly connected component) synthesis problem.

First, we define a light extension of the parametric zone graph as follows. The *extended parametric zone graph* of a PTA $\mathcal{A}$ is identical to its parametric zone graph, except that any transition $(\mathbf{s}, e, \mathbf{s}')$ is replaced with $(\mathbf{s}, (e, b), \mathbf{s}')$, where b is a Boolean flag which is true if time can *potentially* elapse between $\mathbf{s}$ and $\mathbf{s}'$. In practice, b can be computed as follows, given $\mathbf{s} = (l, C)$ and edge e:

1. add a fresh extra clock x_0 to the constraint C, i. e. compute $C \wedge x_0 = 0$
2. compute the successor $\mathbf{s}' = (l', C')$ of $(l, C \wedge x_0 = 0)$ via edge e
3. check whether $C' \Rightarrow x_0 = 0$: if so, then $b =$ false; otherwise $b =$ true.

Introducing such a clock is cheap: the check is not expensive, and the extra clock does not impact the size of the parametric zone graph: As mentioned in [WSW+15], introducing the clock x_0 here is different from the approach of introducing an extra clock for non-Zenoness detection [Tri99] as x_0 is 0 in all nodes of the zone graph and can be eliminated from the memory, therefore not requiring more space nor extra states.

In contrast to non-parametric TAs, the flag b does not necessarily mean that time can necessarily elapse for *all* parameter valuations. Consider the example in Fig. 2c. After taking one loop, we have that $x_0 \leq p$: therefore, x_0 is not necessarily 0, and b is true. But consider v such that $v(p) = 0$: then in l_1 time can never elapse.

However, we show in the following lemma that the flag b *does* denote time elapsing for *strictly positive* parameters.

Lemma 3. *Let $(l, C) \overset{e,b}{\Rightarrow} (l', C')$ be a transition of the extended parametric zone graph of a PTA $\mathcal{A}$. Then, for any strictly positive parameter valuation in $C'{\downarrow_P}$, there exists an equivalent transition in $v(\mathcal{A})$ in which time can elapse.*

Proof. First note that, for any $v \models C'{\downarrow_P}$, an equivalent concrete transition exists in $v(\mathcal{A})$, from Lemma 1. Now, since b is true, the extra clock x_0 in the state of the extended parametric zone graph corresponding to (l, C') is either unbounded, or bounded by some parametric linear term *plt*. If it is unbounded, then time can elapse for any valuation, and the lemma holds trivially. Assume $x_0 \leq plt$ for some *plt*. As our parameters are strictly positive, then for any valuation v, $v(plt)$ evaluates to a strictly positive rational, and therefore time can elapse along this transition in $v(\mathcal{A})$. ■

Definition 6. *An infinite symbolic run $\mathbf{r}$ is non-Zeno if all its associated concrete runs are non-Zeno.*

In the remainder of this section, given an edge $e = (l, g, a, R, l')$, $e.R$ denotes that the clocks in R reset along e.

The following theorem states that an infinite symbolic run is non-Zeno iff the time can (potentially) elapse along infinitely many edges and, whenever a clock is bounded from above, then eventually either this clock is reset or it becomes unbounded.

Theorem 2. *Let $\mathbf{r} = \mathbf{s}_0 \overset{(e_0,b_0)}{\Rightarrow} \mathbf{s}_1 \overset{(e_1,b_1)}{\Rightarrow} \cdots$ be an infinite symbolic run of the extended parametric zone graph of a CUB-PTA $\mathcal{A}$. $\mathbf{r}$ is non-Zeno if and only if*

- $*$ *there exist infinitely many k such that $b_k = true$; and*
- $\star$ *for all $x \in X$, for all $i \geq 0$, given $\mathbf{s}_i = (l_i, C_i)$, if $pub(l_i, x) \neq \infty$, there exists j such that $j \geq i$ and $x \in e_j.R$ or $pub(l_j, x) = \infty$.*

Proof. See Appendix D. ■

We now show in the following that synthesizing parameter valuations for which there exists a non-Zeno infinite run reduces to an SCC searching problem.

First, given an SCC *scc*, we denote by *scc*.K the parameter constraint associated with *scc*, i. e. $C{\downarrow_P}$, where (l, C) is any state of the SCC.[7]

[7] Following a well-known result for PTAs, all symbolic states belonging to a same cycle in a parametric zone graph have the same parameter constraint.

Theorem 3. *Let $\mathcal{A}$ be a CUB-PTA of finite extended parametric zone graph $\mathcal{G}$. Let v be a strictly positive parameter valuation. $v(\mathcal{A})$ contains a non-Zeno infinite run if and only if $\mathcal{G}$ contains a reachable SCC scc such that $v \models scc.K$ and*

- † *scc contains a transition $\mathbf{s} \overset{(e,b)}{\Rightarrow} \mathbf{s}'$ such that $b = true$; and*
- ‡ *for every clock x in X, given $\mathbf{s} = (l, C)$, if $pub(l, x) \neq \infty$ for some state $\mathbf{s}$ in scc, there exists a transition in scc with label (e, b) such that $x \in e.R$.*

Proof. See Appendix E. ■

Therefore, from Theorem 3, synthesizing valuations yielding an infinite symbolic run reduces to an SCC searching problem in the extended parametric zone graph. Then, we need to test each SCC against two conditions: whether it contains a transition which can be locally delayed (i. e. whether it contains a transition where $b =$ true); and whether every clock having an upper bound other than ∞ at some state is reset along some transition in the SCC. Then, for all SCCs matching these two conditions, we return the associated parameter constraint.

We give in Algorithm 2 an algorithm synthNZ to solve the non-Zeno synthesis problem for CUB-PTAs. synthNZ simply iterates on the SCCs, and gathers their associated parameter constraints whenever they satisfy the conditions in Theorem 3.

Algorithm 2: CUB-PTA non-Zeno synthesis algorithm synthNZ($\mathcal{A}$)

```
    Input: CUB-PTA $\mathcal{A}$ and its extended parametric zone graph $\mathcal{G}$
    Output: constraint $K_{NZ}$ gathering valuations for which there is a non-Zeno
            infinite run
1   $K_{NZ} \leftarrow \perp$ while there are un-visited states in $\mathcal{G}$ do
2       find a new SCC scc;
3       mark all states in scc as visited;
4       if scc satisfies † and ‡ then
5           $K_{NZ} \leftarrow K_{NZ} \vee scc.K$ ;
6   return $K_{NZ}$;
```

If the extended parametric zone graph $\mathcal{G}$ is finite, then the correctness and completeness of synthNZ immediately follow from Theorem 3. If only an incomplete part of $\mathcal{G}$ is computed (e. g. by bounding the exploration depth, or the number of explored states, or the execution time) then only the $\Leftarrow$ direction of Theorem 3 holds: in that case, the result of synthNZ is correct but non-complete, i. e. it is a valid under-approximation. In the context of parametric model checking, knowing which parameter valuations violate the property is already very helpful to the designer, as it helps to discard unsafe valuations, and to refine the model.

6 Experiments

6.1 Environment

We implemented our algorithms in the latest version of IMITATOR [AFKS12].[8] The Parma Polyhedra Library (PPL) [BHZ08] is integrated inside the core of IMITATOR in order to solve mainly linear inequality system problems. Experiments were run on an Intel Core 2 Duo P8600 at 2.4 GHz and 4 GiB of memory.

6.2 Experiments

We compare three approaches:

1. A cycle detection synthesis without the non-Zenoness assumption (called synthCycle). The result may be an over-approximation of the actual result, as some of the parameters synthesized may yield only Zeno cycles. If synthCycle does not terminate, its result is an under-approximation of an over-approximation, therefore considered as potentially invalid; that is, there is no guarantee of correctness for the synthesized constraint.
2. Our CUB-detection (Algorithm 1) followed by synthesis (Algorithm 2): the result may be under-approximated, as only the valuations for which the PTA is CUB are considered.
3. Our CUB-transformation (CUBtrans in Algorithm 3) followed by synthesis (Algorithm 2) on the resulting disjunctive CUB-PTA. If the algorithm terminates, then the result is exact, otherwise it may be under-approximated.

We use a modified version of the Tarjan algorithm for detecting SCCs.

We consider various benchmarks to compare our techniques: protocols (CSMA/CD, Fischer [AHV93], RCP, WFAS), hardware circuits (And-Or, flip-flop), scheduling problems (Sched5), a networked automation system (simop) and various academic benchmarks.

We give from left to right in **??** the case study name and its number of clocks, parameters and locations. For synthCycle, we give the computation time (TO denotes a time-out at 3600 s), the constraint type ($\bot$, $\top$ or another constraint) and the validity of the result: if synthCycle terminates, the result is an over-approximation, otherwise it is potentially invalid. For CUBdetect (resp. CUBtrans) we give the detection (resp. transformation) time, the total time (including synthNZ), the result, and whether it is an under-approximation or an exact result. We also mention whether CUBdetect outputs that all, none or some valuations make the PTA CUB; and we give the number of locations in the transformed disjunctive CUB-PTA output by CUBtrans. The percentage is used to compare the number of valuations (comparison obtained by discretization) output by the three algorithms, with CUBtrans as the basis (as the result is exact).

The toy benchmark CUBPTA1 is a good illustration: CUBtrans terminates after 0.073 s (and therefore its result is exact) with some constraint. CUBdetect

[8] For experimental data including source and binary, see `imitator.fr/static/NFM17`

is faster (0.015 s) but infers that only some valuations are CUB and analyzes only these valuations; the synthesized result is only 69 % of the expected result. In contrast, synthCycle is much faster (0.006 s) but obtains too many valuations (208 % of the expected result) as it infers many Zeno valuations.

Interpretation of the experiments Let us discuss the results. First, synthCycle almost always outputs a possibly invalid result (neither an under- nor an over-approximation), which justifies the need for techniques handling non-Zeno assumptions. In only one case (CUBPTA1), it outputs a non-trivial over-approximation. In two cases, it happens to give an exact answer, as the over-approximation of $\perp$ necessarily means that $\perp$ is the exact result. In contrast, CUBtrans gives an exact result in five cases, a non-trivial under-approximation in two cases; the five remaining cases are a disappointing result in which $\perp$ is output as an under-approximation. By studying the model manually, we realized that some non-Zeno cycles actually exist for some valuations, but our synthesis algorithm was not able to derive them. Only in one of these cases (Sched5), synthCycle outputs a more interesting result than CUBtrans.

The transformation is relatively reasonable both in terms of added locations (in the worst case, there are 40 instead of 10 locations, hence four times more, for WFAS) and in terms of transformation time (the worst case is 1.2 s for Sched5). Our experiments do not allow us to fairly compare the time of synthCycle (without non-Zenoness) and synthNZ (with non-Zenoness assumption) as, without surprise due to the undecidability of synthesis, most analyses do not terminate. Only two benchmarks terminate for both algorithms, but are not significant (< 1 s).

Note that flip-flop is a hardware circuit modeled using a bi-bounded inertial delay, and is therefore CUB for all valuations.

An interesting benchmark is WFAS, for which our transformation procedure terminates whereas synthCycle does not. Therefore, we get an exact result while the traditional procedure cannot produce any valuable output.

As a conclusion, CUBdetect seems to be faster but less complete than CUBtrans. As for CUBtrans, its result is almost always more valuable than synthCycle, and therefore is the most interesting algorithm.

7 Conclusion

We proposed a technique to synthesize valuations for which there exists a non-Zeno infinite run in a PTA. By adding accepting states, this allows for parametric model checking with non-Zenoness assumption. Our techniques rely on a transformation to a disjunctive CUB-PTA (or in some cases on a simple detection of the valuation for which the PTA is already CUB), and then on a dedicated cycle synthesis algorithm. We implemented our techniques in IMITATOR and compared our algorithms on a set of benchmarks.

Future works Our technique relying on CUB-PTAs extends the technique of CUB-TAs: this technique is shown in [WSW+15] to be the most efficient for performing non-Zeno model checking for TAs. However, for PTAs, other techniques (such as yet to be defined parametric extensions of strongly non-Zeno TAs [TYB05] or guessing zone graph [HSW12]) could turn more efficient and should be investigated. Studying whether other techniques can be proposed is therefore on our agenda.

In addition, parametric stateful timed CSP (PSTCSP) [ALSD14] is a formalism for which the CUB assumption seems to be natively verified. Therefore, studying non-Zeno parametric model checking for PSTCSP, as well as transforming PTAs into PSTCSP models, would be an interesting direction of research.

Studying the decidability of the underlying decision problem should be done for famous subclasses of PTAs constraining the use of parameters (namely L/U-PTAs, L-PTAs and U-PTAs [HRSV02]) as well as for new semantic subclasses that we recently proposed and that benefit from decidability results (namely integer-point PTAs and reset-PTAs [ALR16]).

An interesting future will be to design a multi-core extension of our non-Zeno synthesis algorithm; this could be done by reusing parallel depth first search algorithms for finding cycles [ELPvdP12].

Finally, combining our synthesis algorithms with IC3 [CGMT13], as well as extending them to hybrid systems [SÁC+15] is also of high practical interest.

References

ABB+16. Lăcrămioara Aştefănoaei, Saddek Bensalem, Marius Bozga, Chih-Hong Cheng, and Harald Ruess. Compositional parameter synthesis. volume 9995 of *Lecture Notes in Computer Science*, pages 60–68, 2016.

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, 2009.

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

AFKS12. Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36, 2012.

AHV93. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

ALR16. Étienne André, Didier Lime, and Olivier H. Roux. Decision problems for parametric timed automata. In *ICFEM*, volume 10009 of *Lecture Notes in Computer Science*, pages 400–416. Springer, 2016.

ALSD14. Étienne André, Yang Liu, Jun Sun, and Jin Song Dong. Parameter synthesis for hierarchical concurrent real-time systems. *Real-Time Systems*, 50(5-6):620–679, 2014.

AM15. Étienne André and Nicolas Markey. Language preservation problems in parametric timed automata. In *FORMATS*, volume 9268 of *Lecture Notes in Computer Science*, pages 27–43. Springer, 2015.

And15. Étienne André. What's decidable about parametric timed automata? In *FTSCS*, Communications in Computer and Information Science, pages 52–68. Springer, 2015.

BBLS15. Nikola Beneš, Peter Bezděk, Kim G. Larsen, and Jiří Srba. Language emptiness of continuous-time parametric timed automata. In *ICALP, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2015.

BDM$^+$98. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *CAV*, volume 1427 of *LNCS*, pages 546–550. Springer, 1998.

BG06. H. Bowman and R. Gómez. How to stop time stopping. *Formal Aspects of Computing*, 18(4):459–493, 2006.

BHZ08. Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

BLN03. D. Beyer, C. Lewerentz, and A. Noack. Rabbit: A Tool for BDD-based Verification of Real-Time Systems. In *CAV*, pages 122–125. Springer, 2003.

CGMT13. Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Parameter synthesis with IC3. In *FMCAD*, pages 165–168. IEEE, 2013.

DHQ$^+$08. Jin Song Dong, Ping Hao, Shengchao Qin, Jun Sun, and Wang Yi. Timed automata patterns. *IEEE Transactions on Software Engineering*, 34(6):844–859, 2008.

ELPvdP12. Sami Evangelista, Alfons Laarman, Laure Petrucci, and Jaco van de Pol. Improved multi-core nested depth-first search. In *ATVA*, volume 7561 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2012.

GB07. R. Gómez and H. Bowman. Efficient detection of Zeno runs in timed automata. In *FORMATS*, volume 4763 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2007.

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.

HSW12. Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed Büchi automata. *Formal Methods in System Design*, 40(2):122–146, 2012.

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

KMH01. Lina Khatib, Nicola Muscettola, and Klaus Havelund. Mapping temporal planning constraints into timed automata. In *TIME*, pages 21–27. IEEE Computer Society, 2001.

KP12. Michał Knapik and Wojciech Penczek. Bounded model checking for parametric timed automata. *Transactions on Petri Nets and Other Models of Concurrency*, 5:141–159, 2012.

LPY97. Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2), 1997.

Min67. Marvin L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Inc., 1967.

SÁC+15. Stefan Schupp, Erika Ábrahám, Xin Chen, Ibtissem Ben Makhlouf, Goran Frehse, Sriram Sankaranarayanan, and Stefan Kowalewski. Current challenges in the verification of hybrid systems. In *CyPhy*, volume 9361 of *Lecture Notes in Computer Science*, pages 8–24. Springer, 2015.

SLDP09. Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards flexible verification under fairness. In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.

Tri99. Stavros Tripakis. Verifying progress in timed systems. In *AMAST*, pages 299–314, 1999.

TYB05. Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed Büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.

Wan02. F. Wang. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram. In *Formal Techniques for Networked and Distributed Systems*, pages 235–250. Springer, 2002.

WSW+15. Ting Wang, Jun Sun, Xinyu Wang, Yang Liu, Yuanjie Si, Jin Song Dong, Xiaohu Yang, and Xiaohong Li. A systematic study on explicit-state non-Zenoness checking for timed automata. *IEEE Transactions on Software Engineering*, 41(1):3–18, 2015.

A Proof of Theorem 1

Theorem 1 (recalled). *The non-Zeno emptiness problem is undecidable for PTAs with at least four clocks and one (bounded) parameter.*

Proof. By reduction from the halting problem of a deterministic 2-counter-machine, which is undecidable [Min67]. Let us encode a 2-counter machine (2CM) using PTAs. Several encodings were proposed in the literature. We rely here on an encoding proposed in [AM15], that requires one single (bounded) parameter p and four clocks. The proof of Theorem 1 does not require to modify the 2CM encoding of [AM15]; still, we recall it in Appendix B for sake of completeness. Basically, that encoding is such that a special location $l_{\texttt{halt}}$ in the PTA is reachable iff the 2-counter machine halts. This encoding has the specificity that the unique parameter p is used not only to denote the maximum possible value of the 2 counters (which is often the case in PTA-based encodings in the literature), but also to bound the number of operations of the 2CM that the PTA can simulate. That is, for any valuation v of p, the encoding of the 2CM will stop after $v(p)$ steps.

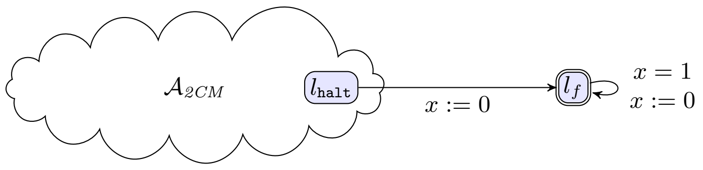


Fig. 3: Undecidability of the non-Zeno emptiness problem

Let $\mathcal{A}_{2CM}$ denote the PTA encoding a given 2-counter machine using the encoding of [AM15]. We extend $\mathcal{A}_{2CM}$ as shown in Fig. 3: from the location encoding the halting location (i. e. $l_{\texttt{halt}}$), we add a transition to a new location l_f. Then, this location has a self-loop guarded with $x = 1$ and resetting x, where x is any of the four clocks used in the encoding in [AM15].

First, recall from [AM15] that $l_{\texttt{halt}}$ is reachable iff the 2-counter-machine halts, and for parameter valuations v such that $v(p)$ is larger than or equal to the maximum value of the counters along the (unique) run of the machine. Then, once $l_{\texttt{halt}}$ is reached, l_f is reachable without condition. Then, once l_f is reached, it can be visited infinitely often every 1 time unit, thanks to the guard and the reset of x. Hence, once l_f is reached, there exists an infinite non-Zeno run for any parameter valuation for which l_f is reachable. However, if the 2CM does not halt, l_f is not reachable; furthermore, thanks to the encoding of [AM15], for any valuation, any run stops after at most $v(p)$ discrete steps, and therefore there is no infinite non-Zeno run for any valuation. Hence, there exists an infinite non-Zeno run iff the 2-counter-machine halts. ∎

B An existing 2-counter machine encoding

For sake of self-containment, we recall in the following the 2-counter machine encoding of [AM15, Theorem 12].

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 $t = 0$. Finally, clock z is used to count the number of steps of the two-counter machine: this is where this construction differs from the classical ones (e. g. [AHV93,JLR15,BBLS15]), 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.

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. 4). This way, exactly one time unit elapses in this module, and clock x_i is decreased by p, 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.

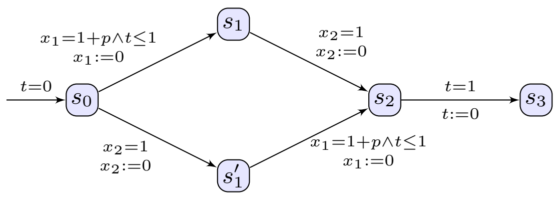


Fig. 4: Encoding incrementation with a rational parameter

We wish to forbid the infinite non-halting run in the machine. For this, we add a third counter, which will be incremented every other step of the resulting three-counter machine. This is achieved by adding a fourth clock z that is reset when $z = 1 - p$, and therefore its value is "incremented" at each gadget. In addition, we add a guard $z \neq 0$ to any transition leading the the initial location of a gadget: therefore, after exactly p steps, we will have that $z = 0 \wedge t = 0$, and the PTA encoding of the machine will be blocked. We then have the property that if $\mathcal{M}$ does not halt, the simulation in the associated PTA will be finite.

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.

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

C An algorithm to transform a PTA into a CUB-PTA

Let us now show that any PTA can be transformed into a disjunctive CUB-PTA. A PTA is not a CUB-PTA if there exists an edge entailing a decreasing upper bound for some clock: this is "problematic" edge. A problematic edge $e = (l, g, a, R, l')$ is detected by comparing the upper bounds of location l, guard g and location l'. Unfortunately, when these upper bounds contain parameters, their comparison might be impossible, so all cases have to be considered. Since all problematic edges can be detected, any PTA can easily be transformed into a disjunctive CUB-PTA by splitting locations and enforcing their invariants to cater for all possible cases. A new initial location is created, that is connected to these copies. Thus, all problematic edges are removed, and the transformed model is a disjunctive CUB-PTA.

Algorithm 3 presents the transformation of a PTA into a disjunctive CUB-PTA. It comprises five parts preceded by an initialization phase. Note that, in order to reduce the state space explosion and to improve efficiency, the algorithm generates only the necessary parameter relations and operates on-the-fly:

0. Initially, the automaton considered is the input one, and an empty set of constraints is associated with each location, that will be modified by the algorithm. The pair composed of automaton and constraints is added to a queue of elements to be handled until all of them have been considered;
1. The first part (lines 4–33) splits the different constraints cases for the problematic edges. For each of these, K (resp. θ) gathers the parametric (resp. timed) constraint w.r.t. the edge.
2. In lines 35–38, a location is created for each constraint in $constraints_q(l)$. Its associated invariant is a conjunction of K and $I(l)$ in L' and $cub\_ls(l)$.
3. Then, these new edges are created to connect these new locations (lines 39–42) ;
4. Finally, in lines 43–47, all problematic edges are deleted.
5. At the end of the loop (line 48), $\mathcal{A}_q$ is a CUB-PTA that is added to the disjunctive CUB-PTA $\mathcal{D}$. When exiting the loop, the initial state l_0 is linked to all initial states of CUB-PTAs before $\mathcal{D}$ is returned.

Algorithm 3: $\mathsf{CUBtrans}(\mathcal{A})$: Transformation into a CUB-PTA

```
Input: PTA $\mathcal{A} = (\Sigma, L, l_0, X, P, K_0, I, E)$
Output: PTA $\mathcal{D}$ associated with a disjunctive CUB-PTA
$\mathcal{A}_0 \leftarrow \mathcal{A}$;  $\forall l \in L, constraints_0(l) \leftarrow \emptyset$;
$queue\ \mathcal{Q} \leftarrow \langle \mathcal{A}_0, constraints_0 \rangle$;
while $\mathcal{Q} \neq \emptyset$ do
    $\langle \mathcal{A}_q(\Sigma, L', l'_0, X, P, K'_0, I, E'), constraints_q \rangle \leftarrow dequeue(\mathcal{Q})$;
    $Kadding \leftarrow \top$;
    while $Kadding = \top$ do
        $Kadding \leftarrow \bot$;
        foreach $edge\ (l, g, a, R, l') \in E'$ do
            $K \leftarrow \bigwedge_{x \in X} \big[pub(I(l), x) \leq pub(g, x) \wedge$
                    if $x \notin R$ then $pub(I(l), x) \leq pub(I(l'), x)$ else $\top\big]$;
            $\theta \leftarrow \bigwedge_{x \in X} \big[(I(l), x) \wedge (g, x) \wedge$ if $x \notin R$ then $(I(l'), x)$ else $\top\big]$;
            if $\theta \notin constraints_q(l)$ then
                if $K\ is\ \bot$ then
                    $constraints_q(l) \leftarrow constraints_q(l) \cup \{\theta\}$;
                if $K\ is\ a\ parameter\ constraint\ and\ K \wedge K'_0 \neq \emptyset$ then
                    if $(\neg K \wedge K'_0) \neq \emptyset$ then
                        $\mathcal{A}' \leftarrow (\Sigma, L', l'_0, X, P, (\neg K \wedge K'_0), I, E')$;
                        $\mathcal{Q} \leftarrow \langle \mathcal{A}', constraints_q(l) \cup \{\theta\} \rangle$;
                    $K'_0 \leftarrow K'_0 \wedge K$;
            foreach $constraint\ K\ in\ constraints_q(l)$ do
                $K' \leftarrow \bigwedge_{x \in X} \big[pubI(l), x) \leq pub(g, x) \wedge$
                        if $x \notin R$ then $pub(I(l), x) \leq pub(k, x)$ else $\top\big]$;
                $\theta' \leftarrow \bigwedge_{x \in X} \big[(I(l), x) \wedge (g, x) \wedge$ if $x \notin R$ then $(K, x)$ else $\top\big]$;
                if $\theta' \notin constraints_q(l)$ then
                    if $K'\ is\ \bot$ then
                        $constraints_q(l) \leftarrow constraints_q(l) \cup \{\theta'\}$;
                    if $K'\ is\ a\ parameter\ constraint\ and\ K' \wedge K'_0 \neq \emptyset$ then
                        if $(\neg K' \wedge K'_0) \neq \emptyset$ then
                            $\mathcal{A}' \leftarrow (\Sigma, L', l'_0, X, P, (\neg K' \wedge K'_0), I, E')$;
                            $\mathcal{Q} \leftarrow \langle \mathcal{A}', constraints_q(l) \cup \{\theta'\} \rangle$;
                        $K'_0 \leftarrow K'_0 \wedge K'$;
            if $constraints_q(l)\ is\ increased$ then
                $Kadding \leftarrow \top$;
    $cub\_ls \leftarrow \emptyset$;
    foreach $constraint\ K\ in\ constraints_q(l)$ do
        create a new location $cub\_l$;
        $I(cub\_l) \leftarrow K \wedge I(l)$;  $L' \leftarrow L' \cup \{cub\_l\}$;
        $cub\_ls \leftarrow cub\_ls \cup \{cub\_l\}$;
    foreach $location\ cub\_l\ in\ cub\_ls$ do
        foreach $edge\ (l', g, a, R, l)\ or\ (l, g, a, R, l')\ respectively\ with\ l' \in L'$ do
            $E' \leftarrow (l', g, a, R, cub\_l)$ or $(cub\_l, g, a, R, l')$;
            $E' \leftarrow (cub\_l', g, a, R, cub\_l)$ or $(cub\_l, g, a, R, cub\_l')$ with $cub\_l' \in cub\_ls$;
    foreach $edge\ e(l, g, a, R, l') \in E'$ do
        $K \leftarrow \bigwedge_{x \in X} \big[pub(I(l), x) \leq pub(g, x) \wedge$
                if $x \notin R$ then $pub(I(l), x) \leq pub(I(l'), x)$ else $\top\big]$;
        if $K = \bot\ or\ (K\ is\ a\ parameter\ constraint\ and\ K \wedge K'_0 = \emptyset)$ then
            $E' \leftarrow E' \setminus \{e\}$
    $\mathcal{D} \cup \mathcal{A}_q$;
foreach $\mathcal{A}_q(\Sigma, L', l'_0, X, P, K'_0, I, E') \in \mathcal{D}$ do
    add edge from $l_0$ of $\mathcal{D}$ to set of $cub\_ls \in l'_0 \cup l'_0$ with $K'_0$ as guard;
return $\mathcal{D}$;
```

D Proof of Theorem 2

Theorem 2 (recalled). *Let* $\mathbf{r} = \mathbf{s}_0 \overset{(e_0,b_0)}{\Rightarrow} \mathbf{s}_1 \overset{(e_1,b_1)}{\Rightarrow} \cdots$ *be an infinite symbolic run of the extended parametric zone graph of a CUB-PTA* $\mathcal{A}$*.* $\mathbf{r}$ *is non-Zeno if and only if*

- $*$ *there exist infinitely many* k *such that* $b_k = \mathit{true}$*; and*
- $\star$ *for all* $x \in X$*, for all* $i \geq 0$*, given* $\mathbf{s}_i = (l_i, C_i)$*, if* $\mathit{pub}(l_i, x) \neq \infty$*, there exists* j *such that* $j \geq i$ *and* $x \in e_j.R$ *or* $\mathit{pub}(l_j, x) = \infty$*.*

Proof. $\Rightarrow$ If $\mathbf{r}$ is non-Zeno, $*$ is trivially true. Consider the case when a clock x is bounded from above (i. e. with a parametric upper bound other than ∞): for any parameter valuation, x is bounded from above (even for arbitrarily large valuations). Therefore the clock x must be reset later, or the upper bound becomes infinity since by definition its value goes unbounded along the run; otherwise, we have an empty symbolic state and thus an infeasible run. Hence, $\star$ holds.

$\Leftarrow$ In the following, we show that if $*$ and $\star$ are true, then $\mathbf{r}$ is non-Zeno. Let the following be a segment of $\mathbf{r}$ according to $*$ and $\star$:

$$(l_i, C_i) \overset{(e_i,b_i)}{\Rightarrow} (l_{i+1}, C_{i+1}) \overset{(e_{i+1},b_{i+1})}{\Rightarrow} \cdots (l_j, C_j) \overset{(e_j,b_j)}{\Rightarrow} \cdots (l_k, C_k) \overset{(e_k,b_k)}{\Rightarrow} \cdots$$

where $i \leq j \leq k$ and $b_j = \text{true}$. Furthermore, for all $x \in X$, if $\mathit{pub}(l_j, x) \neq \infty$, there exists m, n such that $i-1 \leq m < j \leq n \leq k-1$ such that $x \in e_m.R$ (or $m = -1$, i. e. x is "reset" before the initial state as all clocks are initially 0) and, $x \in e_n.R$ or $\mathit{pub}(l_n, x) = \infty$ (from condition $\star$). That is, the segment contains a transition which can be delayed locally. Furthermore, the segment covers the "life-span" (between two resets) of all clocks in l_j which have an upper bound other than ∞.

The infinite symbolic run $\mathbf{r}$ is progressive if and only if it takes an unbounded amount of time for any parameter valuation. Since there are infinitely many segments as above in $\mathbf{r}$, if any such segment can take a positive amount of time for all its valuations, then the run $\mathbf{r}$ is progressive and thus non-Zeno, and then non-Zeno for any valuation (from Definition 6). Next, we show that the segment can take a positive amount of time.

Note that because b_j is true, then from Lemma 3, for any strictly positive valuation in $C_n{\downarrow_P}$, the time than *can* elapse from l_j to l_{j+1} is strictly positive. Furthermore, since $\mathcal{A}$ is a CUB-PTA, therefore for any strictly positive valuation $v \models C_n{\downarrow_P}$, from Lemma 2 $v(\mathcal{A})$ is a CUB-TA. From the syntax of CUB-TAs, a run cannot be blocked in the future due to a too small clock upper bound, as clock upper bounds are always non-decreasing. Therefore, for any strictly positive valuation in $C_n{\downarrow_P}$, the parametric time than can elapse from l_m to l_n is strictly positive. Therefore, for any strictly positive valuation $v \models C_n{\downarrow_P}$, there exists an equivalent concrete run in $v(\mathcal{A})$ that is progressive. Hence $\mathbf{r}$ is progressive. With the arguments above, we conclude that the theorem holds.

■

E Proof of Theorem 3

Theorem 3 (recalled). *Let $\mathcal{A}$ be a CUB-PTA of finite extended parametric zone graph $\mathcal{G}$. Let v be a strictly positive parameter valuation. $v(\mathcal{A})$ contains a non-Zeno infinite run if and only if $\mathcal{G}$ contains a reachable SCC scc such that $v \models scc.K$ and*

† *scc contains a transition $\mathbf{s} \overset{(e,b)}{\Rightarrow} \mathbf{s}'$ such that $b = true$; and*
‡ *for every clock x in X, given $\mathbf{s} = (l, C)$, if $pub(l, x) \neq \infty$ for some state $\mathbf{s}$ in scc, there exists a transition in scc with label (e, b) such that $x \in e.R$.*

Proof. $\Rightarrow$ Assume $v(\mathcal{A})$ contains a non-Zeno infinite run. From Lemma 1, there exists an equivalent symbolic run $\mathbf{r}$ in $\mathcal{G}$. Since $\mathcal{G}$ is finite-state, $\mathbf{r}$ must visit a set of states and transitions, denoted as *Inf*, infinitely often. There must be an SCC, say *scc*, which contains *Inf*. *Inf* must contain a transition with a label b being true (by contradiction) and therefore † is trivially true.
Next, we prove ‡ by contradiction. Assume there is a state $\mathbf{s}$ in *scc* where a clock x has an upper bound *plt* which is not ∞ and there is no transition in *scc* which resets x. Because the upper bound of x never decreases (by definition of CUB-PTAs), the upper bound of x at every state in *scc* must be equal to *plt*. Since *scc* contains *Inf*, this implies that $\mathbf{r}$ is Zeno as x is always bounded from above and never reset, which contradicts our assumption that $\mathbf{r}$ is non-Zeno. Thus, *scc* must satisfy ‡.

$\Leftarrow$ Assume there is an SCC in $\mathcal{G}$ satisfying † and ‡. Let $\mathbf{r}$ be a symbolic run which visits every state/transition in the SCC infinitely often. It is easy to see that $\mathbf{r}$ satisfies $*$ of Theorem 2 because of †. By ‡, we conclude that every clock which has an upper bound other than ∞ at a state is reset later. Therefore, $\mathbf{r}$ is non-Zeno by Theorem 2. From Definition 6, any concrete run associated with $\mathbf{r}$ is non-Zeno; from Lemma 1, the parameter valuations having such an equivalent run are exactly $scc.K$. Therefore, we conclude that the theorem holds.

■