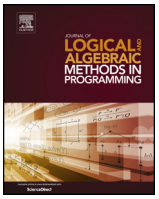
Journal of Logical and Algebraic Methods in Programming 99 (2018) 114–142

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



On clock-aware LTL parameter synthesis of timed automata

Peter Bezděk *, Nikola Beneš, Ivana Černá, Jiří Barnat [1]

Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic

ARTICLE INFO

Article history:
Received 27 February 2017
Received in revised form 9 March 2018
Accepted 17 May 2018
Available online 21 May 2018

Keywords:
Parameter synthesis
Parametric timed automaton
Linear temporal logic
Clock-aware linear temporal logic

ABSTRACT

The parameter synthesis problem for timed automata is undecidable in general even for very simple reachability properties. In this paper we introduce restrictions on parameter valuations under which the parameter synthesis problem is decidable for Clock-Aware LTL properties. The investigated bounded integer parameter synthesis problem could be solved using an explicit enumeration of all possible parameter valuations. We propose an alternative symbolic zone-based method for this problem which can result in a faster computation. Our technique adapts the ideas of the automata-based approach to Clock-Aware LTL model checking of timed automata. In order to simplify the explanation of our method, we first introduce a parameter synthesis algorithm for timed automata, then we describe method for checking Clock-Aware LTL properties of timed automata and finally we combine these two methods together to provide general parameter synthesis algorithm for Clock-Aware LTL properties. To demonstrate the usefulness of our approach, we provide experimental evaluation and compare the proposed method with the explicit enumeration technique.

1. Introduction

Model checking [1] is a formal verification technique applied to check for logical correctness of discrete distributed systems. While it is often used to prove the unreachability of a bad state (such as an assertion violation in a piece of code), with a proper specification formalism, such as the *Linear Temporal Logic* (LTL), it can also check for many interesting liveness properties of systems, such as repeated guaranteed response, eventual stability, live-lock, etc.

Timed automata have been introduced in [2] and have emerged as a useful formalism for modelling time-critical systems as found in many embedded and cyber-physical systems. The formalism is built on top of the standard finite automata enriched with a set of real-time clocks and allowing the system actions to be guarded with respect to the clock valuations. In the general case, such a timed system exhibits infinite-state semantics (the clock domains are continuous). Nevertheless, when the guards are limited to comparing clock values with integers only, there exists a bisimilar finite state representation of the original infinite-state real-time system referred to as the region abstraction [3]. A practically efficient abstraction of the infinite-state space came with the so called zones [4]. The zone-based abstraction is much coarser and the number of zones *reachable* from the initial state can be significantly smaller. This in turn allows for an efficient implementation of verification tools for timed automata, see e.g. UPPAAL [5].

* Corresponding author.
E-mail addresses: bezdek@mail.muni.cz (P. Bezděk), xbenes3@fi.muni.cz (N. Beneš), cerna@fi.muni.cz (I. Černá), barnat@fi.muni.cz (J. Barnat).

[1] This work has been partially supported by the Czech Science Foundation grant No. 18-02177S.

https://doi.org/10.1016/j.jlamp.2018.05.004
2352-2208/

Very often the correctness of a time-critical system relates to proper timing, i.e. it does not only depend on the logical result of the computation, but also on the time at which the results are produced. To that end the designers are not only in the need of tools to verify correctness once the system is fully designed, but also in the need of tools that would help them derive proper time parameters of individual system actions that would make the system as a whole satisfy the required specification. After all this problem of *parameter synthesis* is more urgent in practice than the verification as such.

Related work. The problem of the existence of a parameter valuation for a reachability property of a parametric timed automaton in continuous time has been shown to be undecidable in [6,7] for a parametric timed automaton with as few as 3 clocks. This problem remains undecidable even for integer-valued parameters [8]. A partial solution for the parameter synthesis problem with reachability properties is presented in [9] where the authors provide a semi-decision algorithm which is not guaranteed to terminate in all cases. The authors also introduce a subclass of parametric timed automata, called L/U automata for which the emptiness problem is decidable. Decidability results for the class of L/U automata are further extended in [10]. In particular, the authors show that the emptiness, finiteness and universality problems of the set of parameter valuations for which there is an infinite accepting run are decidable. L/U automata introduce a significant structural restriction to the model of timed automata whereas our goal is to provide a parameter synthesis method for timed automata without any structural restrictions.

To obtain a decidable version of the parameter synthesis problem for timed automata it is sufficient to restrict parameter valuations to bounded integers. When modelling a real-time system, designers can usually provide practical bounds on the time parameters of individual system actions. Therefore, introducing a parameter synthesis method with such a restriction is still reasonable. In [11] the authors show that the problem of the existence of a bounded integer parameter value such that a given property is satisfied is PSPACE-complete for a significant number of properties, which include Timed Computational Tree Logic. They give symbolic algorithms for reachability and unavoidability properties only. While we work with the same model and restrictions as the authors in [11], the fundamental difference is in the supported class of properties where we allow the usage of the whole Clock-Aware LTL (CA-LTL) [12]. This usage of different class of properties requires us to apply a different approach.

Sometimes designers already have at least one correct instance of a parametrised system and they need to synthesise only the parameter valuations that are somewhat similar to the given one. For the given parametrised system and the parameter valuation v_0, the *inverse problem* [13] calls for the computation of a set of parameter valuations such that each valuation v from this set has the same behaviour (in terms of sets of traces) as the given valuation v_0. Algorithms solving the inverse problem are implemented in the tool IMITATOR [13]. Unlike the solution of the inverse problem, our technique targets the search for all correct parameter valuations without any previous knowledge of a correct system behaviour.

There is a plethora of derivatives of linear temporal logics for the specification of properties of real-time systems, timed automata in particular. To name at least some of them, we list TPTL [14], MTL [15], MITL [16], RTTL [17], XCTL [18], CLTL [19], LTLC [20], and CA-LTL. These logics employ various ways of expressing time aspects of underlying systems including one global time clock, time-bounded temporal operators, timing variables with quantifiers, and freeze operators. Some logics are defined with the use of time sampling semantics, which has been shown to be counter-intuitive [21]. The key aspect differentiating CA-LTL from the other logics mentioned above is the ability to properly and intuitively reason about clock values in the classical continuous-time semantics while still preserving practical efficiency of the model checking process.

Similar qualities are found in the branching time logic TCTL [22] a subset of which is actually supported by the UPPAAL tool. We stress that CA-LTL is able to reason about values of clocks in timed automata while still being practically simple enough to allow for an efficient model checking procedure as well as parameter synthesis. Note that the inclusion of time-bounded operators, such as the *until* operator of TCTL, would lead to the expressive power of at least MTL, model checking of which is considered computationally infeasible. CA-LTL can thus be seen as a practically motivated extension of LTL, which is powerful enough to express the same properties as can be expressed by the specification language of the world-wide leading timed automata verification tool UPPAAL.

Our main goal is thus to solve the parameter synthesis problem for CA-LTL properties. The corresponding solution is technically complicated. Therefore, to make the solution easier to comprehend, we explain the main ideas on less complex partial problems first, before combining them to solve our main problem. We first describe a solution to the parameter synthesis problem for LTL specifications [23] and then we continue with a description of the CA-LTL model checking technique [12] separately. Finally, we build the technically more complex solution to the parameter synthesis problem for CA-LTL specification using a combination of the previously explained ideas.

Contribution. This publication is an extended version of the SEFM 2016 conference paper [23] and also adapts techniques from [12]. The conference paper [23] solves the parameter synthesis problem for timed automata and LTL properties. The conference paper [12] introduces the logic CA-LTL and presents a CA-LTL model checking algorithm.

The main contribution of this publication is a symbolic algorithm that solves the parameter synthesis problem for timed automata with bounded integer parameters and CA-LTL properties. The algorithm is a novel and non-trivial combination of the parameter synthesis technique from [23] and the CA-LTL model checking from [12]. We introduce a new finite symbolic abstraction of timed automata with bounded integer parameters and provide an algorithm working over this abstraction. We provide experiments demonstrating the strength of our symbolic algorithm which may be faster by orders of magnitude in comparison with a naive solution.

Outline. The rest of the paper is organised as follows. The problem definition is given in Section 2 that also introduces the basic notions. Section 3 focuses on LTL parameter synthesis of timed automata. We define the symbolic semantics and its finite abstraction. We continue with a description of the parameter synthesis algorithm. In Section 4 we introduce the less complex CA-LTL model checking problem. We describe the corresponding challenges and present an appropriate solution. We finally solve the CA-LTL parameter synthesis of timed automata in Section 5. The solution is based on the combination of ideas from Section 3 and Section 4. Section 5 also provides an experimental evaluation of the proposed technique. Finally, Section 6 concludes the paper.

2. Preliminaries and problem statement

In order to state our main problem formally, we need to describe the notion of a parametric timed automaton. We start by describing some basic notation.

Let P be a finite set of *parameters*. An *affine expression* is an expression of the form $z_0 + z_1 p_1 + \ldots + z_n p_n$, where $p_1, \ldots, p_n \in P$ and $z_0, \ldots, z_n \in \mathbb{Z}$. We use $E(P)$ to denote the set of all affine expressions over P. A *parameter valuation* is a function $v : P \to \mathbb{Z}$ which assigns an integer number to each parameter. Let $lb : P \to \mathbb{Z}$ be a lower bound function and $ub : P \to \mathbb{Z}$ be an upper bound function. For an affine expression e, we use $e[v]$ to denote the integer value obtained by replacing each p in e by $v(p)$. We use $max_{lb,ub}(e)$ to denote the maximal value obtained by replacing each p with a positive coefficient in e by $ub(p)$ and replacing each p with a negative coefficient in e by $lb(p)$. We say that the parameter valuation v respects lb and ub if for each $p \in P$ it holds that $lb(p) \leq v(p) \leq ub(p)$. We denote the set of all parameter valuations respecting lb and ub by $Val_{lb,ub}(P)$. In the following, we consider parameter valuations from $Val_{lb,ub}(P)$ for a fixed lb, ub and P if not stated otherwise.

Let X be a finite set of *clocks*. We assume the existence of a special *zero clock*, denoted by x_0, that has always the value 0. A *simple guard* is an expression of the form $x_i - x_j \sim e$ where $x_i, x_j \in X \cup \{x_0\}$, $e \in E(P)$ and $\sim \in \{\leq, <\}$. We use $G_1(X, P)$ to denote the set of all simple guards over the set of clocks X and the set of parameters P. A *non-diagonal simple guard* is a simple guard of the form $x_i - x_j \sim e$ where $x_i, x_j \in X \cup \{x_0\}$, $e \in E(P)$, $\sim \in \{\leq, <\}$, and $x_i = x_0$ or $x_j = x_0$. We use $\overline{G}_1(X, P)$ to denote the set of all simple non-diagonal guards over a set of clocks X and a set of parameters P. A *guard* is a finite conjunction of simple guards. The empty conjunction is denoted by the Boolean constant **tt**. We use $G(X, P)$ to denote the set of all guards over a set of clocks X and a set of parameters P. A *non-diagonal guard* is a guard restricted to non-diagonal simple guards as conjuncts. We also use $\overline{G}(X, P)$ to denote the set of all non-diagonal guards over a set of clocks X and a set of parameters P.

A *clock valuation* is a function $\eta : X \cup \{x_0\} \to \mathbb{R}_{\geq 0}$ assigning non-negative real numbers to each clock such that $\eta(x_0) = 0$. We denote the set of all clock valuations by $Val(X)$ and the valuation that assigns 0 to each clock by $\mathbf{0}$. Let $g \in G(X, P)$ and v be a parameter valuation and η be a clock valuation. Then $g[v, \eta]$ denotes a Boolean value obtained from g by replacing each parameter p with $v(p)$ and each clock x with $\eta(x)$. A pair (v, η) *satisfies* a guard g, denoted by $(v, \eta) \models g$, if $g[v, \eta]$ evaluates to true. The *semantics* of a guard g, denoted by $[\![g]\!]$, is a set of all valuation pairs (v, η) such that $(v, \eta) \models g$. The semantics of a set of guards G is given by $[\![G]\!] = \bigcap_{g \in G} [\![g]\!]$. For a given parameter valuation v we write $[\![g]\!]_v$ for the set of clock valuations $\{\eta \mid (v, \eta) \models g\}$.

We define two operations on clock valuations. Let η be a clock valuation, d a non-negative real number and $R \subseteq X$ a set of clocks. We use $\eta + d$ to denote the clock valuation that adds the delay d to each clock, i.e. $(\eta + d)(x) = \eta(x) + d$ for all $x \in X$. We further use $\eta\langle R \rangle$ to denote the clock valuation that resets clocks from the set R, i.e. $\eta\langle R \rangle(x) = 0$ if $x \in R$, $\eta\langle R \rangle(x) = \eta(x)$ otherwise.

Definition 2.1 *(PTA).* A *parametric timed automaton* (PTA) is a tuple $M = (L, l_0, X, P, \Delta, Inv)$ where

- L is a finite set of locations,
- $l_0 \in L$ is the initial location,
- X is a finite set of clocks,
- P is a finite set of parameters,
- $\Delta \subseteq L \times \overline{G}(X, P) \times 2^X \times L$ is a finite transition relation, and
- $Inv : L \to \overline{G}(X, P)$ is an invariant function.

We use $q \xrightarrow{g,R}_{\Delta} q'$ to denote $(q, g, R, q') \in \Delta$. The semantics of a PTA is given as a labelled transition system. A *labelled transition system* (LTS) over a set of symbols Σ is a triple $(S, s_0, \to)$, where S is a set of states, $s_0 \in S$ is an initial state and $\to \subseteq S \times \Sigma \times S$ is a transition relation. We use $s \xrightarrow{a} s'$ to denote $(s, a, s') \in \to$.

Given a set of parameters P, we say that a function $f : S \to 2^{Val_{lb,ub}(P)}$ is a *monotonic annotation* for an LTS $(S, s_0, \to)$ if for all $s, s' \in S$ such that $s \xrightarrow{a} s'$ it holds that $f(s) \supseteq f(s')$.

In the following, we assume that the invariants contain upper bounds only. Note that this is without loss of generality, as lower bounds may be always moved from invariants to guards of incoming transitions.

Definition 2.2 *(PTA semantics).* Let $M = (L, l_0, X, P, \Delta, Inv)$ be a PTA and v be a parameter valuation. The *semantics* of M under v, denoted by $[\![M]\!]_v$, is an LTS $(\mathbb{S}^c_M, s_0, \to)$ over the set of symbols $\{act, \infty\} \cup \mathbb{R}_{\geq 0}$, where

- $\mathbb{S}^c_M = L \times Val(X) \cup \{(l, \infty) \mid Inv(l) = \mathbf{tt}\}$ is a set of all concrete states,
- $s_0 = (l_0, \mathbf{0})$, where $\mathbf{0}$ is a clock valuation with $\mathbf{0}(x) = 0$ for all x, and
- the transition relation $\to$ is specified for all $(l, \eta), (l', \eta') \in \mathbb{S}^c_M$ as follows:
  - $(l, \eta) \xrightarrow{d} (l', \eta')$ if $l = l'$, $d \in \mathbb{R}_{\geq 0}$, $\eta' = \eta + d$, and $(v, \eta') \models Inv(l')$,
  - $(l, \eta) \xrightarrow{\infty} (l', \eta')$ if $l = l'$, $\eta' = \infty$, and $\forall d \in \mathbb{R}_{\geq 0}\ (v, \eta + d) \models Inv(l')$,
  - $(l, \eta) \xrightarrow{act} (l', \eta')$ if $\exists g, R : l \xrightarrow{g,R}_{\Delta} l'$, $(v, \eta) \models g$, $\eta' = \eta\langle R \rangle$, and $(v, \eta') \models Inv(l')$.

  The first two kinds of transitions are called *delay transitions*, the latter are called *action transitions*.

We write $s_1 \xrightarrow{act}_d s_2$ if there exists $s' \in \mathbb{S}^c_M$ and $r \in \mathbb{R}^{\geq 0}$ such that $s_1 \xrightarrow{act} s' \xrightarrow{r} s_2$. A proper run π of $[\![M]\!]_v$ is an alternating sequence of delay and action transitions that begins with a delay transition $\pi = (l_0, \eta_0) \xrightarrow{d_0} (l_0, \eta_0 + d_0) \xrightarrow{act} (l_1, \eta_1) \xrightarrow{d_1} \cdots$ and is either infinite or ends with a ∞ delay transition. The length of a proper run $|\pi|$ is the number of action transitions it contains. A proper run is called Zeno if the sum of all its delays is finite.

Remark 2.3. Observe that a PTA $M = (L, l_0, X, P, \Delta, Inv)$ can be considered to be a timed automaton (TA) when $P = \emptyset$. In such case the TA semantics is denoted by $[\![M]\!]$ as only one empty parameter valuation is applicable. We can also derive a TA from a PTA by applying a parameter valuation. Let v be a parameter valuation and M a PTA. We define $M|_v$ to be the TA obtained from M by replacing each occurrence of a parameter p with the value $v(p)$. Note the fact that $[\![M|_v]\!] = [\![M]\!]_v$.

We now introduce the standard notion of clock zones of TA [24]. Every zone is described by a set of diagonal constraints of the form $x_i - x_j \prec_{ij} c_{ij}$ where $c_{ij} \in \mathbb{R}$, $\prec_{ij} \in \{<, \leq\}$ for all clocks x_i, $x_j \in X \cup \{x_0\}$. (Recall that x_0 is a special clock that has always the value 0.) We use these standard operations on zones: intersection $Z \cap Z'$, reset $Z\langle R \rangle = \{\eta\langle R \rangle \mid \eta \in Z\}$, and time passing $Z^{\uparrow} = \{\eta + d \mid \eta \in Z, d \in \mathbb{R}_{\geq 0}\}$. The zones may be efficiently represented using difference bound matrices (DBM) [25, 26]. Although there may be different representations of the same zone, it is a standard result that there exists a unique canonical representation in which the bounds $\prec_{ij} c_{ij}$ are as tight as possible.

In order to keep the number of zones finite, we use the standard *k-extrapolation* construction [24,26–28]. Let Z be a zone and let $\prec_{ij} c_{ij}$ be the bounds in its canonical representation. Let $M(x)$ be the highest bound in the guards of TA that contain the clock x. The extrapolated zone $\mathcal{E}(Z)$ is defined by the set of diagonal constraints $x_i - x_j \prec'_{ij} c'_{ij}$ where

$$\prec'_{ij} c'_{ij} = \begin{cases} < \infty & \text{if } c_{ij} > M(x_i), \\ < -M(x_j) & \text{if } c_{ij} < -M(x_j), \\ \prec_{ij} c_{ij} & \text{otherwise.} \end{cases}$$

We now define the syntax and semantics of the clock-aware linear temporal logic. The atomic propositions of this logic are going to be of two kinds—those that consider properties of locations and those that consider properties of clocks. The former ones, which we call location propositions are just arbitrary symbols that are assigned to locations via a labelling function. The latter ones are simple non-diagonal guards over the set of clocks.

Definition 2.4 *(CA-LTL syntax).* Let $Ap = Lp \cup G$ where Lp is a set of *location propositions* and $G \subseteq \overline{G}_1(X, \emptyset)$ is a set of simple non-diagonal guards without parameters. A *clock-aware linear temporal logic* (CA-LTL) formula over Ap is defined as follows:

$$\varphi ::= l \mid g \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\, \mathbf{U}\, \varphi$$

where $l \in Lp$ and $g \in G$.

We also use the standard derived Boolean operators such as $\wedge$ and $\Rightarrow$, and the usual derived temporal operators $\mathbf{F}\varphi \equiv \mathbf{tt}\, \mathbf{U}\, \varphi$, $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$.

We want our semantics of CA-LTL to reason about continuous linear time. We thus need a notion of a (continuous) suffix of a proper run. For a proper run $\pi = (l_0, \eta_0) \xrightarrow{d_0} (l_0, \eta_0 + d_0) \xrightarrow{act} (l_1, \eta_1) \xrightarrow{d_1} (l_1, \eta_1 + d_1) \xrightarrow{act} \cdots$ we define its suffix $\pi^{k,t}$ as follows:

- if $|\pi| > k$ and $t \leq d_k$ then $\pi^{k,t} = (l_k, \eta_k + t) \xrightarrow{d_k - t} (l_k, \eta_k + d_k) \xrightarrow{act} \cdots$,
- if $|\pi| = k$ then $\pi^{k,t} = (l_k, \eta_k + t) \xrightarrow{\infty} (l_k, \infty)$,
- otherwise, $\pi^{k,t}$ is undefined.

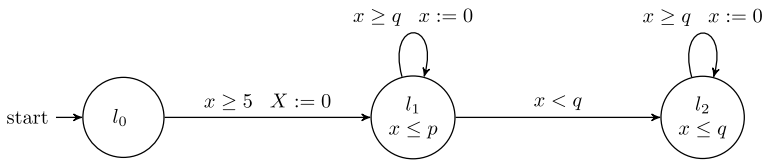


Fig. 1. A sensor system.

Note that the condition $|\pi| = k$ implies that π ends with $\cdots (l_k, \eta_k) \xrightarrow{\infty} (l_k, \infty)$. We further define an ordering on the set of suffixes of π, denoted by $\lhd_\pi$ as follows: $\pi^{i,t} \lhd_\pi \pi^{j,s}$ if both $\pi^{i,t}$ and $\pi^{j,s}$ are defined and either $i < j$ or $i = j$ and $t \leq s$. (The semantics is that $\pi^{i,t}$ is an "earlier" suffix of π than $\pi^{j,s}$.)

Definition 2.5 *(CA-LTL semantics).* Let $\mathcal{L} : L \to 2^{Lp}$ be a function that assigns a set of location propositions to each location. The semantics of a CA-LTL formula φ on a proper run $\pi = (l_0, \eta_0) \xrightarrow{d} (l_0, \eta_0 + d_0) \xrightarrow{act} (l_1, \eta_1) \xrightarrow{d1} \cdots$ with a labelling $\mathcal{L}$ is given as follows:

$$
\begin{aligned}
\pi &\models p && \iff p \in \mathcal{L}(l_0)\\
\pi &\models g && \iff \eta_0 \models g\\
\pi &\models \neg\varphi && \iff \pi \not\models \varphi\\
\pi &\models \varphi \vee \psi && \iff \pi \models \varphi \text{ or } \pi \models \psi\\
\pi &\models \varphi \,\mathbf{U}\, \psi && \iff \exists k, t : \pi^{k,t} \text{ is defined, } \pi^{k,t} \models \psi, \text{ and}\\
& && \quad \forall j, s \text{ such that } \pi^{j,s} \lhd_\pi \pi^{k,t} : \pi^{j,s} \models \varphi \vee \psi
\end{aligned}
$$

For a timed automaton A with a location labelling function $\mathcal{L}$, we say that A with $\mathcal{L}$ satisfies φ, denoted by $(A, \mathcal{L}) \models \varphi$ if for all proper runs π of $[\![A]\!]$, $\pi \models \varphi$. For a parametric timed automaton M with a location labelling function $\mathcal{L}$ and a parameter valuation v, we say that M under v with $\mathcal{L}$ satisfies φ, denoted by $(M, v, \mathcal{L}) \models \varphi$ if $([\![M]\!]_v, \mathcal{L}) \models \varphi$.

Remark 2.6. Note that the standard *linear temporal logic* (LTL) without the *next* temporal operator **X** is a special case of CA-LTL when $G = \emptyset$. The reason for the omission of **X** is mainly the absence of its clear semantics in the context of continuous time. Also note that there are other reasons to omit the **X** operator from LTL such as no clear semantics in concurrent systems with interleaving or the lack of stuttering-invariance (for discussion on **X** see e.g. [29]).

Example 2.7. Fig. 1 illustrates a sensor system example with $X = \{x, y\}$ and $P = \{p, q\}$. The sensor has two modes (standard and regular) and is able to transmit data at a potentially variable rate. The sensor moves from the initial location l_0 to the location l_1 which represents the standard mode after an initialisation phase which takes at least 5 time units. This initial transition resets all clocks in X. The standard mode allows data transmission to occur within $[q, p]$ time units. On the other hand, the regular mode, represented by the location l_2, transmits data regularly every q time units. The mode can only be switched in one direction, from standard to regular. The clock x measures the time elapsed since the last data transmission and the clock y measures the time since the first visit of l_1. Note the fact that for any parameter valuation v where $v(q) > v(p)$ the sensor is forced to switch to the regular mode. On the other hand, for any parameter valuation v where $v(q) \leq v(p)$ the sensor is allowed to stay in the standard mode. We can consider a CA-LTL property $\mathbf{FG}(x \leq 3)$ which states that the data transmission (bound with the reset of x) is going to happen somewhat regularly after an initialisation phase. The satisfaction of this property depends on the constraints restricting the parameter valuations. For example, the property holds when parameter constraints $p \leq 3, q \leq 2$ are considered.

Given a parametric timed automaton M, a labelling function $\mathcal{L}$, and an CA-LTL property φ, *the parameter synthesis problem* is to compute the set of all parameter valuations v such that $(M, v, \mathcal{L}) \models \varphi$. Unfortunately, it is known that the parameter synthesis problem for a PTA is undecidable even for very simple (reachability) properties [6]. Instead of solving the general problem, we thus focus on a more constrained version which is still reasonable for practical purposes.

Now, we state our main problem for two specification languages. Note the fact that the first problem is a special case of the second one.

Problem 2.8 *(The LTL bounded integer parameter synthesis problem).* Given a parametric timed automaton $M = (L, l_0, X, P, \Delta, Inv)$, an LTL property φ, a labelling function $\mathcal{L}$, a lower bound function lb and an upper bound function ub, *the LTL bounded integer parameter synthesis problem* is to compute the set of all parameter valuations v such that $(M, v, \mathcal{L}) \models \varphi$ and $lb(p) \leq v(p) \leq ub(p)$ for each $p \in P$.

Problem 2.9 *(The CA-LTL bounded integer parameter synthesis problem).* Given a parametric timed automaton $M = (L, l_0, X, P, \Delta, Inv)$, an CA-LTL property φ, a labelling function $\mathcal{L}$, a lower bound function lb and an upper bound function ub, *the CA-LTL bounded integer parameter synthesis problem* is to compute the set of all parameter valuations v such that $(M, v, \mathcal{L}) \models \varphi$ and $lb(p) \leq v(p) \leq ub(p)$ for each $p \in P$.

3. LTL parameter synthesis of PTA

The LTL bounded integer parameter synthesis problem (Problem 2.8) is trivially decidable using the standard zone-based abstraction and the explicit enumeration of all parameter valuations. In order to avoid the necessity of the explicit enumeration of all parameter valuations we propose a solution based on a combination of the zone-based abstraction and a symbolic representation of parameter valuation sets. Our algorithmic framework that solves this problem consists of three steps.

As the first step, we apply the standard automata-based LTL model checking of timed automata [2] to parametric timed automata. We employ this approach in the following way. From a PTA M and an LTL formula φ we produce a product parametric timed Büchi automaton (PTBA) A. The accepting runs of the automaton A correspond to the runs of M violating the formula φ.

As the second step, we employ a symbolic semantics of the PTBA A with a suitable extrapolation. From the abstract symbolic state space of the PTBA A we finally produce a Büchi automaton B with a monotonic annotation f in which each state is associated symbolic information about parameter valuations. This transformation is described in Subsection 3.1.

As the last step, we need to detect all parameter valuations for which there exists an accepting v-run of the Büchi automaton B with a monotonic annotation f. To that end, we employ a new algorithm, which we call Cumulative NDFS. The algorithm is described in detail in Subsection 3.2.

We now proceed with the definitions of a Büchi automaton, a parametric timed Büchi automaton and the corresponding semantics.

Definition 3.1 *(BA).* A *Büchi automaton* (BA) is a tuple $B = (Q, q_0, \Sigma, \rightarrow, F)$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, Σ is a finite set of symbols, $\rightarrow \subseteq Q \times \Sigma \times Q$ is a set of transitions, and $F \subseteq Q$ is a set of accepting states (acceptance condition). A run $\pi = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \ldots$ of B is an infinite sequence of states $q_0 q_1 q_2 \ldots$ such that $q_i \xrightarrow{a_i} q_{i+1}$ for all $i \in \mathbb{N}$. We say π is *accepting* if there exist infinitely many $i \in \mathbb{N}$ such that $q_i \in F$. An ω-word $w = a_0 a_1 a_2 \ldots \in \Sigma^\omega$ is accepted by B if there is an accepting run $\pi = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \ldots$.

Given a set of parameters P, we say that a function $f : Q \rightarrow 2^{Val_{lb,ub}(P)}$ is a *monotonic annotation* for a BA B if for all $q, q' \in Q$ such that $q \xrightarrow{a} q'$ it holds that $f(q) \supseteq f(q')$. Given a monotonic annotation f for a BA B, a run $\pi = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \ldots$ of a BA B is called a v-run if $v \in f(q_i)$ for each state q_i.

Definition 3.2 *(PTBA).* A *parametric timed Büchi automaton* (PTBA) is a pair $A = (M, F)$ where $M = (L, l_0, X, P, \Delta, Inv)$ is a PTA, and $F \subseteq L$ is a set of accepting locations.

Zeno runs represent non-realistic behaviours and it is desirable to ignore them in analysis. Therefore, we are interested only in non-Zeno accepting runs of a PTBA. There is a syntactic transformation to the so-called strongly non-Zeno form [30] of a PTBA, which guarantees that each accepting run is non-Zeno. For the rest of the section, we thus assume that there are no Zeno accepting runs in the PTBA.

Definition 3.3 *(PTBA semantics).* Let $A = (M, F)$ be a PTBA and v be a parameter valuation. The *semantics* of A under v, denoted by $[\![A]\!]_v$, is defined as $[\![A]\!]_v = ([\![M]\!]_v, F)$ with $[\![M]\!]_v = (\mathbb{S}^c_M, s_0, \rightarrow)$.

We say a state $s = (l, \eta) \in \mathbb{S}^c_M$ is *accepting* if $l \in F$. A proper run $\pi = s_0 \xrightarrow{d_0} s'_0 \xrightarrow{act} s_1 \xrightarrow{d_1} s'_1 \xrightarrow{act} \ldots$ of $[\![A]\!]_v$ is accepting if there exists an infinite set of indices i such that s_i is accepting.

3.1. Symbolic semantics

In this subsection we show the construction of a finite system which represents the semantics of a given PTBA. First, we describe a parametric extension of the zone abstraction. This extension is based on constrained parametric difference bound matrices, described in [9]. The intuition behind this extension is that we need to distinguish all the different shapes of zones represented by parametric difference bound matrices. This potential variation in zone shape is caused by ambiguity when comparing two parametric expressions. The removal of this ambiguity requires the addition of new parameter constraints to the constraint set context which is tracked together with each parametric zone.

In addition, this abstraction itself does not guarantee finiteness in our setting. To solve this problem we further introduce a finite parametric extrapolation.

3.1.1. Constrained parametric difference bound matrix

A *constraint* is an inequality of the form $e \sim e'$ where $e, e' \in E(P)$ and $\sim \in \{>, \geq, \leq, <\}$. We define $c[v]$ as the Boolean value obtained by replacing each p in c by $v(p)$. A valuation v *satisfies* a constraint c, denoted $v \models c$, if $c[v]$ evaluates to

true. The *semantics* of a constraint c, denoted $[\![c]\!]$, is the set of all valuations that satisfy c. A finite set of constraints C is called a *constraint set*. A valuation *satisfies* a constraint set C if it satisfies each $c \in C$. The *semantics* of a constraint set C is given by $[\![C]\!] = \bigcap_{c \in C} [\![c]\!]$. A constraint set C is *satisfiable* if $[\![C]\!] \neq \emptyset$. A constraint c *covers* a constraint set C, denoted $C \models c$, if $[\![C]\!] \subseteq [\![c]\!]$.

As in [9], we identify the relation symbol $\leq$ with the Boolean value *true* and $<$ with the Boolean value *false*. Then, we treat the Boolean connectives $\wedge$ and $\Rightarrow$ on relation symbols $\leq, <$ as operations with Boolean values. The following table explains this convention in detail.

$\prec_i$	$\prec_j$	$\prec_i \wedge \prec_j$	$\prec_i \Rightarrow \prec_j$
$<$	$<$	$<$	$\leq$
$<$	$\leq$	$<$	$\leq$
$\leq$	$<$	$<$	$<$
$\leq$	$\leq$	$\leq$	$\leq$

We now define the parametric difference bound matrix, the constrained parametric difference bound matrix, several operations on them, and the symbolic semantics of a PTBA.

Definition 3.4. A *parametric difference bound matrix* (PDBM) over P and X is a set D which contains for all $0 \leq i, j \leq |X|$ a simple guard of the form $x_i - x_j \prec_{ij} e_{ij}$ where $x_i, x_j \in X \cup \{x_0\}$, $\prec_{ij} \in \{\leq, <\}$ and $e_{ij} \in E(P) \cup \{\infty\}$ and $i = j \Rightarrow e_{ii} = 0$. We denote by D_{ij} a simple guard of the form $x_i - x_j \prec_{ij} e_{ij}$ contained in D. Given a parameter valuation v, the *semantics* of D is given by $[\![D]\!]_v = [\![\bigwedge_{i,j} D_{ij}]\!]_v$. A PDBM D is *satisfiable* with respect to v if $[\![D]\!]_v$ is non-empty.

Definition 3.5. A *constrained parametric difference bound matrix* (CPDBM) is a pair (C, D), where C is a constraint set and D is a PDBM and for each $0 \leq i \leq |X|$ it holds that $C \models e_{0i} \leq 0$. The *semantics* of (C, D) is given by $[\![C, D]\!] = \{(v, \eta) \mid v \in [\![C]\!] \wedge \eta \in [\![D]\!]_v\}$. We call (C, D) *satisfiable* if $[\![C, D]\!]$ is non-empty. A CPDBM (C, D) is said to be *in the canonical form* if and only if for all i, j, k, $C \models e_{ij}(\prec_{ik} \wedge \prec_{kj}) e_{ik} + e_{kj}$.

3.1.2. Resetting a clock

Suppose (C, D) is a CPDBM in the canonical form. The reset of the clock x_r in (C, D), denoted by $(C, D)\langle x_r \rangle$, is given as $(C, D\langle x_r \rangle)$ where:

$$D\langle x_r \rangle_{ij} = \begin{cases} D_{0j} & \text{if } i \neq j \text{ and } i = r, \\ D_{i0} & \text{if } i \neq j \text{ and } j = r, \\ D_{ij} & \text{else.} \end{cases}$$

We generalise this definition to a set of clocks: $(C, D)\langle \{x_{i_0}, x_{i_1}, \ldots, x_{i_k}\} \rangle \overset{def}{\Leftrightarrow} (C, D)\langle x_{i_0} \rangle \langle x_{i_1} \rangle \ldots \langle x_{i_k} \rangle$.

3.1.3. Applying a guard

Suppose g is a simple guard of the form $x_i - x_j \prec e$, (C, D) is a CPDBM in the canonical form and $D_{ij} = (e_{ij}, \prec_{ij})$. The application of the simple guard g on (C, D) generally results in a set of CPDBMs and is defined as follows:

$$(C, D)[g] = \begin{cases} \{(C, D[g])\} & \text{if } C \models \neg(e_{ij}(\prec_{ij} \Rightarrow \prec) e), \\ \{(C, D)\} & \text{if } C \models e_{ij}(\prec_{ij} \Rightarrow \prec) e, \\ \{(C \cup \{e_{ij}(\prec_{ij} \Rightarrow \prec) e\}, D), & \text{otherwise,} \\ (C \cup \{\neg e_{ij}(\prec_{ij} \Rightarrow \prec) e\}, D[g])\} & \end{cases}$$

where $D[g]$ is defined as follows:

$$D[g]_{kl} = \begin{cases} (e, \prec) & \text{if } k = i \text{ and } l = j, \\ D_{kl} & \text{else.} \end{cases}$$

We generalise this definition to guards (conjunctions of simple guards) as follows: $(C, D)[g_{i_0} \wedge g_{i_1} \wedge \ldots \wedge g_{i_k}] \overset{def}{\Leftrightarrow} (C, D)[g_{i_0}][g_{i_1}] \ldots [g_{i_k}]$. Note the fact that a conjunction of guards is also a guard.

Let K be a set of CPDBMs. We generalise this definition to the application of a guard g' on K as follows: $K[g'] \overset{def}{\Leftrightarrow} \bigcup_{(C,D) \in K} (C, D)[g']$.

Example 3.6. Let $X = \{x, y\}$ and $P = \{p, q\}$. We illustrate the operation of a guard application. Consider the guard $x < q$ and the CPDBM $cpdbm_1 = (C_1, D_1) = (\{3 \leq p \leq 4, 2 \leq q \leq 4\}, \{0 \leq x \leq p, 0 \leq y \leq p, 0 \leq y - x \leq 0\})$. In this case, neither

$v(p) < v(q)$ nor $v(p) \geq v(q)$ holds for all the parameter valuations in the constraint set C_1. Therefore, we have to split the constraint set C_1 and consider two results with restricted constraint sets. The result of applying the guard is thus $cpdbm_1[x < q] = \{cpdbm_2, cpdbm_3\}$ where $cpdbm_2 = (C_1 \cup \{p < q\}, D_1)$ and $cpdbm_3 = (C_1 \cup \{p \geq q\}, \{0 \leq x < q, 0 \leq y \leq p, 0 \leq y - x \leq 0\})$.

3.1.4. Time successors

Suppose (C, D) is a CPDBM in the canonical form. The time successor of (C, D), denoted by $(C, D)^{\uparrow}$, represents a CPDBM with all upper bounds on clocks removed and is given as $(C, D^{\uparrow})$ where:

$$D^{\uparrow}_{ij} = \begin{cases} (\infty, <) & \text{if } i \neq 0 \text{ and } j = 0, \\ D_{ij} & \text{else.} \end{cases}$$

3.1.5. Canonisation

The reset and time successor operations preserve the canonical form of a CPDBM. After the application of a guard the CPDBM may no longer be in the canonical form and thus a transformation to the canonical form needs to be performed. However, due to the presence of parameters the standard canonisation process [25] can be ambiguous. The canonisation procedure is therefore extended to cope with this ambiguity. As a consequence, the result of the canonisation is not a single CPDBM, but may generally be a set containing potentially more CPDBMs in the canonical form with mutually disjoint constraint sets.

To canonise the given CPDBM we need to derive the tightest constraint on each clock difference. Deriving the tightest constraint on a clock difference can be seen as finding the shortest path in the graph interpretation of the CPDBM. In [9] the authors implement the canonisation using a nondeterministic extension of the Floyd–Warshall algorithm where on each relaxation a split into two different CPDBMs can occur.

First, we define a relation $\longrightarrow_{FW}$ on constrained parametric difference bound matrices as follows, for all $0 \leq k, i, j \leq |X|$:

- $(k, i, j, C_1, D_1) \longrightarrow_{FW} (k, i, j+1, C_2, D_2)$ if $(C_2, D_2) \in (C_1, D_1)[x_i - x_j(\prec_{ik} \wedge \prec_{kj})e_{ik} + e_{kj}]$
- $(k, i, |X|+1, C_1, D_1) \longrightarrow_{FW} (k, i+1, 0, C_1, D_1)$
- $(k, |X|+1, 0, C_1, D_1) \longrightarrow_{FW} (k+1, 0, 0, C_1, D_1)$

The relation $\longrightarrow_{FW}$ can be seen as a representation of the computation steps of the extended Floyd–Warshall algorithm.

Suppose now (C, D) is a CPDBM. The canonical set of (C, D), denoted as $(C, D)_c$, represents a set of CPDBMs with the tightest constraint on each clock difference in D and is defined as follows:

$$(C, D)_c = \{(C', D') \mid (0, 0, 0, C, D) \longrightarrow^{*}_{FW} (|X|+1, 0, 0, C', D')\}$$

We can combine the application of a guard and the subsequent canonisation into one operation defined as follows: $(C, D)[g]_c \overset{def}{\Leftrightarrow} \bigcup_{(C', D') \in (C, D)[g]} (C', D')_c$.

Example 3.7. Let $X = \{x, y\}$ and $P = \{p, q\}$. Consider $cpdbm_2$ and $cpdbm_3$ from Example 3.6. The canonisation of $cpdbm_2$ returns a singleton set containing the same $cpdbm_2$. After the canonisation of $cpdbm_3 = (C_3, D_3)$ we obtain a singleton set containing $cpdbm_4 = (C_3, \{0 \leq x < q, 0 \leq y < q, 0 \leq y - x \leq 0\})$.

3.1.6. Finite abstraction

The goal of this subsection is to obtain a finite semantics for a PTBA. We start with the definition of the PTBA symbolic semantics. Similarly to the non-parametric case, the symbolic semantics may be infinite. Therefore we continue with the definition of a finite abstraction and a corresponding finite abstract symbolic semantics.

Definition 3.8 *(PTBA symbolic semantics).* Let $A = ((L, l_0, X, P, \Delta, Inv), F)$ be a PTBA. Let lb and ub be a lower bound function and an upper bound function on parameters. The *symbolic semantics* of A with respect to lb and ub is a transition system $(\mathbb{S}_A, \mathbb{S}_{init}, \Longrightarrow)$, denoted as $[\![A]\!]_{lb,ub}$, where

- $\mathbb{S}_A = L \times \{[\![C, D]\!] \mid (C, D) \text{ is a } CPDBM\}$ is the set of all symbolic states,
- the set of initial states $\mathbb{S}_{init} = \{(l_0, [\![C, D]\!]) \mid (C, D) \in (\hat{C}, E^{\uparrow})[Inv(l_0)]_c\}$, where
  - E is a PDBM with $E_{i,j} = (0, \leq)$ for each i, j, and
  - $\hat{C} = \{p \geq lb(p) \mid p \in P\} \cup \{p \leq ub(p) \mid p \in P\}$.
- There is a transition $(l, [\![C, D]\!]) \Longrightarrow (l', [\![C', D']\!])$ if

- $l \xrightarrow{g,R}_{\Delta} l'$ and
- there exist satisfiable CPDBMs $(C'', D''), (C', D')$ such that:
  - $(C'', D'') \in (C, D)[g]_c$ and
  - $(C', D') \in (C'', D''\langle R\rangle^{\uparrow})[Inv(l')]_c$.

We say that a state $S = (l, [\![C, D]\!]) \in \mathbb{S}_A$ is accepting if $l \in F$. We say that $\pi = S_0 \Longrightarrow S_1 \Longrightarrow \dots$ is a run of $[\![A]\!]_{lb,ub}$ if $S_0 \in \mathbb{S}_{init}$ and for each $i \in \mathbb{N}_{>0}$ holds $S_i \in \mathbb{S}_A$ and $S_{i-1} \Longrightarrow S_i$. A run π is called a v-run if for each state $S_i = (l_i, [\![C_i, D_i]\!])$ it holds that $v \in [\![C_i]\!]$. A run π is accepting if there exists an infinite set of indices i such that S_i is accepting. For the rest of the paper we fix lb, ub and use $[\![A]\!]$ to denote $[\![A]\!]_{lb,ub}$.

In order to obtain a finite transition system we need to apply a finite abstraction over $[\![A]\!]$. In the standard case of timed automata without parameters we use one of the extrapolation techniques [28,31]. In our parametric setup we define a new finite abstraction called the *pk-extrapolation* which is a parametric extension of the widely used *k-extrapolation* [28]. The k-extrapolation identifies states which are identical except for the clock values that exceed the maximal constant from guards and invariants.

In our parametric setup, we need to define the maximal constant with which each clock within a PTA is compared. We define $M_A(x)$ as the maximal value in $\{max_{lb,ub}(e) \mid e$ is compared with x in a guard or an invariant of the PTA $A\}$. If A is clear from the context we use $M(x)$ instead of $M_A(x)$. The core idea of pk-extrapolation is the same as the idea of k-extrapolation. We substitute each bound on clock difference in the CPDBM whenever this bound exceeds the maximal constant. The precise description of this substitution process is given in the Definition 3.9. Contrary to the non-parametric case, due to the occurrence of parameters in the CPDBM bounds, the substitution process may be ambiguous. In these situations we restrict the parameter values in order to obtain an unambiguous situation. This solution is similar to the constraint set splitting that is done in the application of a guard and in the canonisation procedure. Therefore, the result of pk-extrapolation is a set of CPDBMs instead of a single CPDBM.

Definition 3.9 *(pk-extrapolation).* Let A be a PTBA, $(l, [\![C, D]\!])$ be a symbolic state of $[\![A]\!]$ and $D_{ij} = x_i - x_j \prec_{ij} e_{ij}$ for each $0 \le i, j \le |X|$. We define the *pk-extrapolation* α_{pk} in the following way: $\alpha_{pk}(l, [\![C, D]\!])$ is the set of all $(l, [\![C \cup C', D']\!])$ such that $C' = \bigcup_{0 \le i,j \le |X|} C'_{ij}$, $(C \cup C', D')$ is satisfiable, and for each $0 \le i, j \le |X|$ one of the following conditions holds:

- $D'_{ij} = x_i - x_j < \infty$ and $C'_{ij} = \{e_{ij} > M(x_i)\}$,
- $D'_{ij} = x_i - x_j \prec_{ij} e_{ij}$ and $C'_{ij} = \{-M(x_j) \le e_{ij}, e_{ij} \le M(x_i)\}$,
- $D'_{ij} = x_i - x_j < -M(x_j)$ and $C'_{ij} = \{e_{ij} < -M(x_j)\}$.

Example 3.10. Consider $X = \{x, y\}$, $P = \{p, q\}$, $M(x) = M(y) = 5$, and the symbolic state $(l_1, [\![C, D]\!])$ where $C = \{3 \le p \le 4, 2 \le q \le 4, q \le p\}$ and $D = \{0 \le x \le p, 2q \le y < \infty, 2q \le y - x \le 2p\}$. Note the fact that $2p > 5$ holds for any considered parameter valuation. Now, the pk-extrapolation of $(l_1, [\![C, D]\!])$ results in a set containing the following two symbolic states: $(l_1, [\![C \cup \{5 < 2q\}, \{0 \le x \le p, 5 < y < \infty, 5 < y - x < \infty\}]\!])$ and $(l_1, [\![C \cup \{2q \le 5\}, \{0 \le x \le p, 2q \le y < \infty, 2q \le y - x < \infty\}]\!])$.

Let A be a PTBA. In the following, we write $s_1 \in_v S_2$ if a concrete state s_1 is contained in a symbolic state S_2; more precisely if $s_1 = (l_1, \eta)$ is a concrete state from $[\![A]\!]_v$, $S_2 = (l_2, [\![C, D]\!])$ is a symbolic state from $[\![A]\!]$, $l_1 = l_2$, $v \in [\![C]\!]$, and $\eta \in [\![D]\!]_v$.

Definition 3.11 *(Time-abstracting simulation).* Given an LTS $(S, s_0, \to)$, a time-abstracting simulation R over S is a binary relation satisfying the following conditions:

- $s_1 R s_2$ and $s_1 \xrightarrow{act} s'_1$ implies the existence of $s_2 \xrightarrow{act} s'_2$ such that $s'_1 R s'_2$, and
- $s_1 R s_2$ and $d_1 \in \mathbb{R}^{\ge 0}$ and $s_1 \xrightarrow{d_1} s'_1$ implies the existence of $d_2 \in \mathbb{R}^{\ge 0}$ and $s_2 \xrightarrow{d_2} s'_2$ such that $s'_1 R s'_2$.

We define the largest simulation relation over S ($\preccurlyeq_S$) in the following way: $s \preccurlyeq_S s'$ if there exists a time-abstracting simulation R with $(s, s') \in R$. When S is clear from the context we shall only use $\preccurlyeq$ instead of $\preccurlyeq_S$ in the following.

Definition 3.12 *(PTBA abstract symbolic semantics).* Let $A = (M, F)$ be a PTBA. An *abstraction over* $[\![A]\!] = (\mathbb{S}_A, \mathbb{S}_{init}, \Longrightarrow)$ is a mapping $\alpha : \mathbb{S}_A \to 2^{\mathbb{S}_A}$ such that the following conditions hold:

- $(l', [\![C', D']\!]) \in \alpha(l, [\![C, D]\!])$ implies $l = l' \wedge [\![C']\!] \subseteq [\![C]\!] \wedge [\![C', D]\!] \subseteq [\![C', D']\!]$,
- for each $v \in [\![C]\!]$ there exist S_1, S_2 such that $S_2 = (l, [\![C', D']\!]) \in \alpha(S_1)$ and for each $s \in_v S_2$ there exists a state $s' \in_v S_1$ satisfying $s \preccurlyeq s'$.

An abstraction α is called *finite* if its image is finite. An abstraction α over $[\![A]\!]$ induces a new transition system $[\![A]\!]^\alpha = (\mathbb{Q}_A, \mathbb{Q}_{init}, \Longrightarrow^\alpha)$ where

- $\mathbb{Q}_A = \{S \mid S \in \alpha(S') \text{ and } S' \in \mathbb{S}_A\}$,
- $\mathbb{Q}_{init} = \{S \mid S \in \alpha(S') \text{ and } S' \in \mathbb{S}_{init}\}$, and
- $Q \Longrightarrow^\alpha Q'$ if there is $S \in \mathbb{S}_A$ such that $Q' \in \alpha(S)$ and $Q \Longrightarrow S$.

An accepting state, a run, a v-run, and an accepting run are defined analogously as in the $[\![A]\!]$ case. We say $[\![A]\!]^\alpha$ preserves accepting runs of $[\![A]\!]_v$ for a parameter valuation v if following holds: there exists an accepting v-run of $[\![A]\!]^\alpha$ if and only if there exists an accepting proper run of $[\![A]\!]_v$. If α is finite then $[\![A]\!]^\alpha$ can be viewed as a Büchi automaton.

Example 3.13. We illustrate PTBA abstract symbolic semantics on the sensor PTA from Example 2.7. Consider $X = \{x, y\}$, $P = \{p, q\}$, $M(x) = M(y) = 5$ and the initial parameter bounds $3 \le p \le 4, 2 \le q \le 4$. We perform several transitions in the PTBA abstract symbolic semantics.

The initial state is $s_0 = (l_0, [\![C_0, D_0]\!]) = (l_0, [\![\{3 \le p \le 4, 2 \le q \le 4\}, \{0 \le x < \infty, 0 \le y < \infty, 0 \le y - x \le 0\}]\!])$. When the execution of the PTA transition from l_0 to l_1 is considered in the state s_0, we obtain one successor $s_1 = (l_1, [\![C_0, \{0 \le x \le p, 0 \le y \le p, 0 \le y - x \le 0\}]\!])$. When the execution of the PTA loop transition from l_1 to l_1 is considered in the state s_1, we obtain one successor $s_2 = (l_1, [\![C_0 \cup \{q \le p\}, \{0 \le x \le p, q \le y < \infty, q \le y - x \le p\}]\!])$. When the execution of the PTA loop transition from l_1 to l_1 is considered in the state s_2, we obtain two successors $s_3 = (l_1, [\![C_0 \cup \{q \le p, 5 < 2q\}, \{0 \le x \le p, 5 < y < \infty, 5 < y - x < \infty\}]\!])$ and $s_4 = (l_1, [\![C_0 \cup \{q \le p, 2q \le 5\}, \{0 \le x \le p, 2q \le y < \infty, 2q \le y - x < \infty\}]\!])$. Observe that the Example 3.10 describes the pk-extrapolation applied during the generation of the states s_3 and s_4. When the execution of the PTA transition from l_1 to l_2 is considered in the state s_2, we obtain two successors $s_5 = (l_2, [\![C_0 \cup \{q \le p, 5 < p + q\}, \{0 \le x \le q, q \le y < \infty, q \le y - x \le p\}]\!])$ and $s_6 = (l_2, [\![C_0 \cup \{q \le p, p + q \le 5\}, \{0 \le x \le q, q \le y \le p + q, q \le y - x \le p\}]\!])$.

Observe the fact that the PTBA abstract symbolic semantics has the following property of monotonicity: the set of parameter valuations associated with each state can not grow along any run. Lemma 3.14 states this observation formally. This property follows from the definition of successors in the PTBA abstract symbolic semantics and the definition of operations on CPDBMs.

Lemma 3.14. *Let A be a PTBA, α be an abstraction and $S = (l, [\![C, D]\!])$ be a state in $[\![A]\!]^\alpha$. For every state $S' = (l', [\![C', D']\!])$ reachable from S it holds that $[\![C]\!] \supseteq [\![C']\!]$.*

Theorem 3.15. *Let A be a PTBA. The pk-extrapolation is a finite abstraction such that $[\![A]\!]^{\text{pk-extrapolation}}$ preserves all accepting runs of $[\![A]\!]_v$ for each parameter valuation v.*

The full technically detailed proof of Theorem 3.15 is presented in Appendix A.

We now consider a monotonic annotation f such that $f((l, [\![C, D]\!])) = [\![C]\!]$ for each state $(l, [\![C, D]\!])$ of $[\![A]\!]^{\text{pk-extrapolation}}$. Then, the abstract symbolic semantics of a PTBA A under pk-extrapolation can be viewed as a BA with a monotonic annotation f. Finally, we have a monotonic annotation together with a BA which preserves all accepting runs of $[\![A]\!]_v$ for each parameter valuation v.

3.2. Parameter synthesis algorithm

We recall that our main objective in this section is to find all parameter valuations for which the parametric timed automaton satisfies its LTL specification. So far we have described the automata-based method employed under a parametric setup which produces a BA with a monotonic annotation. We now present our parameter synthesis algorithm for a general BA B with a monotonic annotation f on the input.

The standard automata-based LTL model checking checks the emptiness of the produced Büchi automaton. The emptiness check can be performed using the Nested Depth First Search (NDFS) algorithm [32]. The NDFS algorithm is a modification of the depth first search algorithm which allows a detection of an accepting cycle in the given Büchi automaton.

Contrary to the standard LTL model checking, it is not enough to check the emptiness of the produced Büchi automaton. Our objective is to check the emptiness of the produced Büchi automaton for each considered parameter valuation. To solve this objective, we introduce a new algorithm called the Cumulative NDFS (CNDFS) algorithm which is an extension of the NDFS algorithm.

For the rest of the paper we use $s.[\![C]\!]$ to denote the set $f(s)$ for each state s of the input BA B. Let $c = s_1 \Longrightarrow s_2 \Longrightarrow \ldots \Longrightarrow s_n \Longrightarrow s_1$ be a cycle and let v be a parameter valuation. We say that c is a v-cycle if each state s_i in the c satisfies $v \in s_i.[\![C]\!]$. A cycle c is called accepting if there exists $0 \le i \le n$ such that s_i is accepting. Observe the crucial fact that each state s on a cycle has the same set $s.[\![C]\!]$.

The pseudocode of Cumulative NDFS is given in Algorithm 1. Our modification of NDFS is based on the set *Found* which accumulates each detected parametric valuation v such that an accepting v-cycle was found. In contrast to the NDFS

```
1  Algorithm CumulativeNDFS(B = (S, s_init, Σ, →, Accepting), f)
2      Found ← ∅; Stack ← ∅
3      Outer ← ∅; Inner ← ∅
4      OuterDFS(s_init)
5      return Accepted ← Found
6  Procedure OuterDFS(s)
7      Stack ← Stack ∪ {s}
8      Outer ← Outer ∪ {s}
9      foreach s' such that s → s' do
10         if s' ∉ Outer ∧ s' ∉ Stack ∧ s'.[[C]] ⊈ Found then
11             OuterDFS(s')
12     if s ∈ Accepting ∧ s.[[C]] ⊈ Found then
13         InnerDFS(s)
14     Stack ← Stack \ {s}
15 Procedure InnerDFS(s)
16     Inner ← Inner ∪ {s}
17     foreach s' such that s → s' do
18         if s' ∈ Stack then
19             "Cycle detected"
20             Found ← Found ∪ s'.[[C]]
21             return
22         if s' ∉ Inner ∧ s'.[[C]] ⊈ Found then
23             InnerDFS(s')
```

Algorithm 1: Cumulative NDFS.

algorithm, whenever Cumulative NDFS detects an accepting cycle, corresponding parameter valuations are saved to the set *Found* and the computation continues with a search for another accepting cycle. Note the fact that whenever we reach a state s' with $s'.[\![C]\!] \subseteq Found$ we already have found an accepting v-cycle for all valuations v from $s'.[\![C]\!]$ and there is no need to continue with the search from s' thanks to the monotonicity property. Therefore, we are able to speed up the computation whenever we reach such a state.

Theorem 3.16. *Let B be a BA with a monotonic annotation f. A parameter valuation v is contained in the output of the CumulativeNDFS(B, f) if and only if there exists an accepting v-run of B.*

The proof of this theorem is given in Subsection 3.3.

As the last step in the solution to the parameter synthesis problem, we need to complement the set *Accepted*. Thus, the solution is the set $Val_{lb,ub}(X, P) \setminus Accepted$. To conclude this section, we state that Theorem 3.16 together with Theorem 3.15 imply the correctness of our solution.

3.3. Proof of Theorem 3.16

Lemma 3.17. *If the valuation v is added to the set Found then v is returned by the algorithm in the set Accepted.*

Proof. This follows from the fact that the set *Found* is never decreased and at the end of computation it is assigned to *Accepted*. □

Lemma 3.18. *Let v be a parameter valuation. Let B be a BA with a monotonic annotation f and q be a state of B that does not appear on any v-cycle. The OuterDFS procedure will backtrack from q only after every reachable state s such that $v \in s.[\![C]\!]$ is already backtracked or $s.[\![C]\!] \subseteq Found$.*

Proof. Consider an arbitrary state s such that s is reachable from q. At the time of backtracking from q there are two cases:

- Every path from q to the state s contains a state s' such that $s'.[\![C]\!] \subseteq Found$. The fact that s is reachable from s' implies $s.[\![C]\!] \subseteq s'.[\![C]\!]$ (using Lemma 3.14). Hence, $s.[\![C]\!] \subseteq Found$.
- There exists a path from q to s such that for every state s' on that path it holds that $s'.[\![C]\!] \nsubseteq Found$. In this case, the *OuterDFS* procedure has visited the state s with the state q on the stack. Hence, the *OuterDFS* procedure backtracks from the state q after backtracking from s. □

Lemma 3.19. *Let B be a BA with a monotonic annotation f. For every parameter valuation v, the Cumulative NDFS algorithm returns the set Accepted containing the valuation v if and only if the given B contains an accepting v-cycle c.*

Proof. Whenever the algorithm returns a set *Accepted* containing v there exists an accepting v-cycle c. Such an accepting cycle can be constructed using *OuterDFS* and *InnerDFS* search stack at the time of adding the valuation v to the set *Found*.

The difficult case is to show that whenever there exists an accepting v-cycle in the given graph then the algorithm returns a set *Accepted* containing v. Suppose an accepting v-cycle exists in the given graph and the algorithm returns a set *Accepted* such that $v \notin$ *Accepted*. Note that this means that all states s' reachable from the initial state with $v \in s'.[\![C]\!]$ are visited by the *OuterDFS* procedure.

Let q be the first accepting state on a v-cycle from which *InnerDFS* is started. There are two cases:

- There exists a path from a state q to some state on the stack of *OuterDFS* and each state s on the path is unvisited by *InnerDFS* and $s.[\![C]\!] \not\subseteq$ *Found* at the time of starting *InnerDFS* from q.
- For all paths from a state q to some state p on the stack of *OuterDFS* there exists a state s on the path such that $s.[\![C]\!] \subseteq$ *Found* or s is a state already visited by *InnerDFS*.

For the first case the algorithm will detect an accepting cycle as expected and will add the valuation $v \in q.[\![C]\!]$ to the set *Found*. From Lemma 3.17 we get $v \in$ *Accepted* and we have reached a contradiction with the assumption $v \notin$ *Accepted*.

For the second case, whenever the path $q \rightsquigarrow p$ contains a state s such that $v \in s.[\![C]\!] \subseteq$ *Found* we reach contradiction (using Lemma 3.17). Let r be the first visited state that is reached from q during *InnerDFS* and is on a cycle through q. Let q' be an accepting state that started *InnerDFS* in which r was visited for the first time. Notice the fact that *InnerDFS* was started from q' before starting from q. There are two cases:

- The state q' is reachable from q. Then there is an accepting cycle $c' = q' \rightsquigarrow r \rightsquigarrow q \rightsquigarrow q'$. If c' contains a state s such that $v \in s.[\![C]\!] \subseteq$ *Found* we reach a contradiction using Lemma 3.17. Suppose there is no state s with $v \in s.[\![C]\!] \subseteq$ *Found* on the cycle c'. The cycle c' was not found previously. However, this contradicts our assumption that q is the first accepting state from which we missed a cycle.
- The state q' is not reachable from q. Notice the fact that $v \in q'.[\![C]\!]$ (this follows from Lemma 3.14) and therefore every cycle containing the state q' is a v-cycle. If q' appears on a cycle, then an accepting v-cycle was missed before starting *InnerDFS* from q, contrary to our assumption. If q' does not appear on a cycle then by Lemma 3.18 we backtracked from q in the *OuterDFS* before backtracking from q' and therefore *InnerDFS* started from q before starting from q'. We have reached a contradiction with the fact that *InnerDFS* started from q' before starting from q. □

Lemma 3.20. *The CumulativeNDFS algorithm always terminates.*

Proof. From the fact that the number of vertices is finite we get that the size of the sets *Inner* and *Outer* is bounded. Each invocation of *InnerDFS* (*OuterDFS*) procedure increases the size of the set *Inner* (*Outer*). Hence, the *CumulativeNDFS* algorithm cannot proceed infinitely due to the upper bound on the size of the set *Inner* and *Outer*. □

Theorem *(Theorem 3.16). Let B be a BA with a monotonic annotation f. A parameter valuation v is contained in the output of the CumulativeNDFS(B, f) if and only if there exists an accepting v-run of B.*

Proof. By Lemma 3.20 the algorithm is guaranteed to terminate returning the set *Accepted*. The partial correctness, the $\Rightarrow$ case: By Lemma 3.19 for each $v \in$ *Accepted* there exists an accepting v-cycle and for each $v \notin$ *Accepted* there is no accepting v-cycle. The partial correctness, the $\Leftarrow$ case: Analogously. □

3.4. State space storage

One of the performance critical parts of our parameter synthesis algorithm is the work with the state space storage. We use the state space storage to look up and store information about the presence of each state in the sets *Inner*, *Outer*, and *Stack*. We refer to this information as *data*. In this section we are working with a BA state space representing the abstract symbolic semantics of PTBA. Therefore we are allowed to reference the l, C, D components of each state.

A straightforward implementation would simply store each state together with its data. Such a solution is only efficient when a unique representation of states is available. Without such a unique representation the storage operations have to perform expensive equivalence checks with each stored state in the worst case scenario. In [11] the authors introduce unique representation based on the computation of an integer hull. The integer hull of a given set is the convex hull of all integer elements of the set.

The solution of [11] assumes the existence of an upper bound for each clock. We do not have such an upper-bound assumption and therefore this solution is not directly applicable in our technique. However, we use the integer hull as a heuristic approximation of a unique representation of a CPDBM instead. This way we obtain a practically efficient solution that deals with the non-existence of a unique representation of a state.

The solution is based on two mappings. The first mapping, denoted by M_1 maps a given integer hull to a list of CPDBM representations. Each such list contains the representations of semantically different CPDBMs with the same integer hull.

Thanks to M_1 we can quickly distinguish states with different integer hulls. However, each storage operation still needs to perform the expensive computation of the integer hull. In order to reduce number of the integer hull computations, we introduce the second mapping, denoted by M_2. This second mapping serves as a cache which maps a given CPDBM to its unique representative in the storage. Once a CPDBM representative is resolved, it is saved in M_2.

```
1  Procedure InitializeStorage()
2      Storage ← ∅; M1 ← ∅; M2 ← ∅
3  Procedure SetData(l, C, D, data)
4      if M2(C, D) ≠ ∅ then
5          (C′, D′) ← M2(C, D)
6          Storage(l, C′, D′) ← data
7      else
8          IH ← integerHull(C, D)
9          foreach (C′, D′) in M1(IH) do
10             if ⟦C′, D′⟧ = ⟦C, D⟧ then
11                 M2(C, D) ← (C′, D′)
12                 Storage(l, C′, D′) ← data
13                 return
14         M2(C, D) ← (C, D)
15         M1(IH) ← M1(IH) ∪ {(C, D)}
16         Storage(l, C, D) ← data
17 Procedure GetData(l, C, D)
18     if M2(C, D) ≠ ∅ then
19         (C′, D′) ← M2(C, D)
20         return Storage(l, C′, D′)
21     else
22         IH ← integerHull(C, D)
23         foreach (C′, D′) in M1(IH) do
24             if ⟦C′, D′⟧ = ⟦C, D⟧ then
25                 M2(C, D) ← (C′, D′)
26                 return Storage(l, C′, D′)
27         M2(C, D) ← (C, D)
28         M1(IH) ← M1(IH) ∪ {(C, D)}
29         Storage(l, C, D) ← initialData
30         return initialData
```

Algorithm 2: State space storage operations.

The pseudocode of the state space storage operations is given in Algorithm 2. Note that the procedures *SetData* and *GetData* are analogous. These two mappings as well as the storage itself can be implemented using hash tables.

4. Clock-aware LTL model checking of TA

In order to solve Problem 2.9 we first describe a solution to the corresponding CA-LTL model checking problem which is less complicated as no parameters are involved. We show why standard timed Büchi automaton emptiness checking can not be used to solve this problem. Then we continue with a description of the solution from [12] which is based on zone abstraction and ultraregions.

First we use an example of a timed automaton and some CA-LTL formulae to explain the intricacies of the CA-LTL model checking problem.

Example 4.1. Let us consider a timed automaton as given in Fig. 2 with the labelling function $\mathcal{L}$ assigning to each location its own name only, i.e. $\mathcal{L}(l) = \{l\}$ for all l. Let us further consider the CA-LTL formulae

$$\varphi = \mathbf{G}(l_1 \Rightarrow ((x \leq 3 \wedge y \leq 3)\,\mathbf{U}\,(x > 3 \wedge y > 3))) \quad \text{and} \quad \psi = \mathbf{F}\, l_3.$$

Note that while there exists a run satisfying φ and a run satisfying ψ, there is no run satisfying their conjunction, $\varphi \wedge \psi$. The reason is that the runs satisfying φ always perform the reset of y at time 0, while the runs satisfying ψ always perform the reset of y at some other time, to be able to satisfy the guard $y < 6$ together with $x = 6$.

First of all, note that there is no obvious way of combining this TA with a Büchi automaton representing the formula φ (or its negation). The reason is that while staying in l_0, the satisfaction of the guards $x \leq 3$, $y \leq 3$ changes. We could try splitting each location into several ones such that staying in each of these new locations ensures no changes of the guards. However, under the standard TA semantics, such feature is impossible. Indeed, if there were two locations with invariants

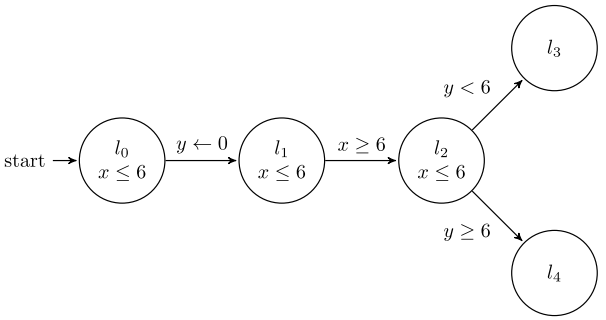


Fig. 2. Timed automaton $A_{4.1}$.

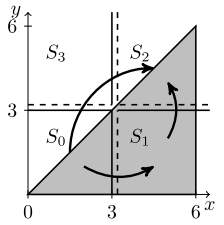


Fig. 3. Illustration of the need to consider diagonals separately.

$x \leq 3$ and $x > 3$, respectively, no transition between them could be enabled at any time. There thus appears to be no direct way of reducing our problem to timed Büchi automaton emptiness.

One way of solving the problem whether a timed automaton satisfies a CA-LTL formula is to evaluate the formula as a standard LTL formula over the automaton's region graph. Suppose that we have the standard region graph construction [3], in which the maximal bounds on each clock also include the bounds appearing in the formula. Then the satisfaction of guards inside a region never changes. However, it is well known that the number of regions is impractically large. In the following, we therefore aim to provide a zone-based model checking approach.

Considering the standard zone-based approach [24], the main issue pointed out above remains—the satisfaction of the guards differs for various parts of a zone. Our first idea is to slice (pre-partition) the zones according to the guards of the formula. In our example, this would mean to slice the zones into one of the four "quadrants" $[0,3] \times [0,3]$, $(3,\infty) \times [0,3]$, $(3,\infty) \times (3,\infty)$, $[0,3] \times (3,\infty)$. These are illustrated in Fig. 3 and named S_0 to S_3. Every sliced zone now respects the guards $x \leq 3$, $y \leq 3$. Note, however, that this partitioning also comes with the need of describing new transitions between the newly defined zone slices. These new transitions correspond to the passage of time within the original zone. Now, consider again Example 2 and the zone that is created after the transition from l_0 to l_1 is taken. The zone is defined as the set of all valuations of clocks η such that $\eta(x) - \eta(y) \geq 0$ and $\eta(x), \eta(y) \in [0,6]$, also illustrated with the greyed area in Fig. 3.

Let us take the S_0 slice of this zone. The next slice is not uniquely determined. One candidate is the S_1 slice as all valuations with $\eta(x) - \eta(y) > 0$ will reach this slice with the passage of time. However, we also have another candidate, the S_2 slice. This is due to the fact that all valuations with $\eta(x) - \eta(y) = 0$ reach the S_2 slice immediately after leaving the S_0 slice. We cannot take both options with a nondeterministic choice. This would introduce incorrect behaviour, as then there would be a run in the zone graph satisfying the conjunction of formulae $\varphi \wedge \psi$. Therefore, we also need to take diagonals into account. The problem here is very similar to the problem that led to the inclusion of diagonals into standard region graphs. Our slicing areas thus somehow resemble regions, only much larger. Also, their count is only dependent on the number of guards appearing in the CA-LTL formula, and may thus be expected to be reasonable. For their similarity with regions, we call these areas *ultraregions*.

4.1. Zone-ultraregion semantics

In the following, let X be a fixed set of clocks and G a fixed set of simple non-diagonal guards over X without parameters. For a clock $x \in X$, we define $\mathcal{I}_x$ to be the coarsest interval partition of $\mathbb{R}_{\geq 0}$ that respects the simple guards in G, i.e. all values in an interval have to satisfy the same subset of simple guards of G. Let further B_x denote the set of bounds the clock x is compared against in the simple guards of G and let $B_{x-y} = \{a - b \mid a \in B_x, b \in B_y\}$. Let then $\mathcal{I}_{x-y} = \{(-\infty, c_0), [c_0, c_0], (c_0, c_1), \ldots, (c_k, \infty)\}$ where $B_{x-y} = \{c_0, \ldots, c_k\}$ and $c_0 < c_1 < \cdots < c_k$. For a valuation $\eta \in \mathbb{R}_{\geq 0}^{X \cup P}$ we use $\mathcal{I}_x(\eta)$ to denote the interval of $\mathcal{I}_x$ that contains the value $\eta(x)$, similarly for $\mathcal{I}_{x-y}(\eta)$. We say that $\mathcal{I}_x(\eta)$ is *unbounded* if it is of the form $[c, \infty)$ or (c, ∞), otherwise we say that it is *bounded*. We now define an equivalence relation with respect to a set of simple guards G on clock valuations.

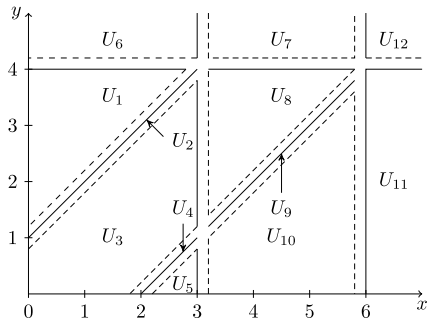


Fig. 4. Ultraregions of $G = \{x \leq 3, x < 6, y \leq 4\}$.

Definition 4.2 *(Ultraregions).* Let X be a set of clocks, G a set of simple non-diagonal guards over X without parameters. We define an equivalence relation $\simeq_G$ on $\mathbb{R}_{\geq 0}^{X \cup P}$ as follows: $\eta \simeq_G \eta'$ if for all x, $\mathcal{I}_x(\eta) = \mathcal{I}_x(\eta')$, and for all y, z such that $\mathcal{I}_y(\eta)$ and $\mathcal{I}_z(\eta)$ are *bounded*, $\mathcal{I}_{y-z}(\eta) = \mathcal{I}_{y-z}(\eta')$. The equivalence classes of $\simeq_G$ are called the *ultraregions* of G.

Note that every ultraregion is uniquely identified by a choice of intervals from $\mathcal{I}_x$ and $\mathcal{I}_{x-y}$ for all clocks x, y. Also note that a choice in $\mathcal{I}_{x-y}$ always determines a choice in $\mathcal{I}_{y-x}$.

Example 4.3. Let $G = \{x \leq 3, x < 6, y \leq 4\}$. Then $\mathcal{I}_x = \{[0, 3], (3, 6), [6, \infty)\}$, $\mathcal{I}_y = \{[0, 4], (4, \infty)\}$, and $\mathcal{I}_{x-y} = \{(-\infty, -1), [-1, -1], (-1, 2), [2, 2], (2, \infty)\}$.

The ultraregions of G look as follows:

$$U_1 = \{\eta = (x, y) \mid x \in [0, 3], y \in [0, 4], x - y < -1\}$$
$$U_2 = \{\eta = (x, y) \mid x \in [0, 3], y \in [0, 4], x - y = -1\}$$
$$U_3 = \{\eta = (x, y) \mid x \in [0, 3], y \in [0, 4], x - y \in (-1, 2)\}$$
$$U_4 = \{\eta = (x, y) \mid x \in [0, 3], y \in [0, 4], x - y = 2\}$$
$$U_5 = \{\eta = (x, y) \mid x \in [0, 3], y \in [0, 4], x - y > 2\}$$
$$U_6 = \{\eta = (x, y) \mid x \in [0, 3], y > 4\}$$
$$U_7 = \{\eta = (x, y) \mid x \in (3, 6), y > 4\}$$
$$U_8 = \{\eta = (x, y) \mid x \in (3, 6), y \in [0, 4], x - y \in (-1, 2)\}$$
$$U_9 = \{\eta = (x, y) \mid x \in (3, 6), y \in [0, 4], x - y = 2\}$$
$$U_{10} = \{\eta = (x, y) \mid x \in (3, 6), y \in [0, 4], x - y > 2\}$$
$$U_{11} = \{\eta = (x, y) \mid x \geq 6, y \in [0, 4]\}$$
$$U_{12} = \{\eta = (x, y) \mid x \geq 6, y > 4\}$$

These ultraregions are illustrated in Fig. 4.

Let $U \neq U'$ be ultraregions. We say that U' is a successor of U if for all $\eta \in U$ there exists $d \in \mathbb{R}_{>0}$ such that $\eta + d \in U'$ and $\forall 0 \leq d' \leq d : \eta + d' \in U \cup U'$.

Lemma 4.4. *An ultraregion has at most one successor.*

This allows us to denote the successor of U by $succ(U)$. If U has no successor, we additionally define $succ(U) = U$. Note that $succ(U) = U$ if and only if all clocks are unbounded in U, i.e. for every $\eta \in U$ and every $d \in \mathbb{R}_{\geq 0}$, $\eta + d \in U$.

Let now $R \subseteq X$ be a set of clocks. The *reset* of U with respect to R, denoted by $U\lfloor R \rfloor$, is defined as follows:

$$U\lfloor R \rfloor = \{U' \mid U' \text{ is an ultraregion } \wedge \exists \eta \in U : \eta\langle R \rangle \in U'\}$$

Example 4.5. Continuing with Example 4.3, we may see that e.g. $succ(U_8) = U_7$, $succ(U_{12}) = U_{12}$, $U_{11}\lfloor x \rfloor = \{U_1, U_2, U_3\}$, $U_9\lfloor x, y \rfloor = \{U_3\}$, and $U_5\lfloor x \rfloor = \{U_3\}$.

Note that the ultraregions are a special case of zones (and the extrapolation does not change them). We may thus also apply the zone operations to ultraregions. However, be aware that the ultraregion reset and the zone reset of an ultraregion are different operations. This is why we use a different notation for $U\lfloor R\rfloor$. We may now define the zone-ultraregion semantics of a timed automaton.

Definition 4.6 *(Zone-ultraregion automaton).* Let $A = (L, l_0, X, \Delta, Inv)$ be a TA and let G be a set of simple non-diagonal guards without parameters. The *zone-ultraregion automaton* (ZURA) of A with respect to G is a labelled transition system whose states are triples (l, Z, U) where $l \in L$, Z is a clock zone, and U is a ultraregion of G.

The initial state is (l_0, Z_0, U_0) where $Z_0 = \{\mathbf{0}\}^{\uparrow} \cap Inv(l_0)$ and U_0 is the ultraregion containing the zero valuation $\mathbf{0}$. The transitions are of two kinds:

- delay transitions: $(l, Z, U) \xrightarrow{\delta} (l, Z, succ(U))$ whenever $Z \cap succ(U) \neq \emptyset$ and $U = succ(U) \Rightarrow Z = Z^{\uparrow}$,
- action transitions: $(l, Z, U) \xrightarrow{act} (l', \mathcal{E}(Z'), U')$ whenever $l \xrightarrow{g,R}_{\Delta} l'$, $U' \in U\lfloor R\rfloor$, $Z' = ((Z \cap U \cap g)\langle R\rangle \cap U')^{\uparrow} \cap Inv(l')$, and $Z' \cap U' \neq \emptyset$.

A combination of a ZURA with a location labelling function $\mathcal{L}$ is interpreted as a Kripke structure [1]. The states and transitions of this Kripke structure are the states and transitions of the ZURA, forgetting the labels of transitions. The state labelling function $\mathcal{L}_K$ is defined as $\mathcal{L}_K(l, Z, U) = \mathcal{L}(l) \cup \{g \in G \mid U \models g\}$. Here, $U \models g$ denotes that all valuations of U satisfy g. Due to the definition of ultraregions, this is equivalent to the existence of a valuation in U satisfying g.

The following theorem gives us a solution to the CA-LTL model checking problem by reducing it to the problem of standard LTL model checking of a Kripke structure.

Theorem 4.7. *[12] Let A be a TA, let A_{ZURA} be its zone-ultraregion automaton with respect to G. Let further φ be a CA-LTL formula over G. Then $A \models \varphi$ iff $A_{ZURA} \models \varphi$.*

Now we finish this section with an example of a zone-ultraregion automaton.

Example 4.8. Continuing with Example 4.1, Fig. 5 represents the ZURA of the timed automaton $A_{4.1}$ with respect to $G = \{x \leq 3, y \leq 3\}$.

5. Clock-aware LTL parameter synthesis of PTA

We have shown in the previous section the reason why Problem 2.9 can not be reduced to parametric timed Büchi automaton emptiness checking in a simple way. Therefore we need to extend the symbolic semantics defined in Section 3 with the concept of ultraregions.

Our parameter synthesis algorithm for a PTA A and a CA-LTL formula φ consists of the following steps.

As the first step, we apply a symbolic semantics with ultraregions and a suitable extrapolation on automaton A. We build a finite labelled transition system A_{sym} with a monotonic annotation f in which each state is associated symbolic information about parameter valuations. This transformation is described in Subsection 5.1.

As the second step, we apply standard automata-based LTL model checking where guards from φ are treated as atomic propositions. We show that such application with CA-LTL formulae is permissible. From A_{sym} and φ we produce a product automaton B in a standard way. On top of that the monotonic annotation f for A_{sym} can be trivially viewed as a monotonic annotation for B where all annotations are taken from the A_{sym} component of the product. Then, for a Büchi automaton B with a monotonic annotation f it holds that the accepting ν-runs of the automaton B correspond to the ν-runs of A_{sym} violating formula φ.

As the last step, we need to detect all parameter valuations for which there exists an accepting ν-run of a Büchi automaton B with a monotonic annotation f. To that end, we employ the Cumulative NDFS algorithm described in Subsection 3.2.

We now continue with the definition of an appropriate symbolic semantics.

5.1. Symbolic semantics

As the first step we define an abstract symbolic semantics based on Definition 3.8 and Definition 4.6. To that end we also need to formally redefine the pk-extrapolation in order to accept symbolic states with a fourth ultraregion component.

Definition 5.1. Let $A = (L, l_0, X, P, \Delta, Inv)$ be a PTA. Let G be a set of simple non-diagonal guards without parameters. Let $(l, [\![C, D]\!], U)$ be a symbolic state from $L \times \{[\![C, D]\!] \mid (C, D)$ is a $CPDBM\} \times \{U \mid U$ is an ultraregion of $G\}$. Let $D_{ij} = x_i - x_j \prec_{ij} e_{ij}$ for each $0 \leq i, j \leq |X|$. We define the *pk-extrapolation* α_{pk} in the following way. $\alpha_{pk}(l, [\![C, D]\!], U)$ is the set of all $(l, [\![C \cup C', D']\!], U)$ such that $C' = \bigcup_{0 \leq i, j \leq |X|} C'_{ij}$ and for each $0 \leq i, j \leq |X|$ one of the following conditions holds:

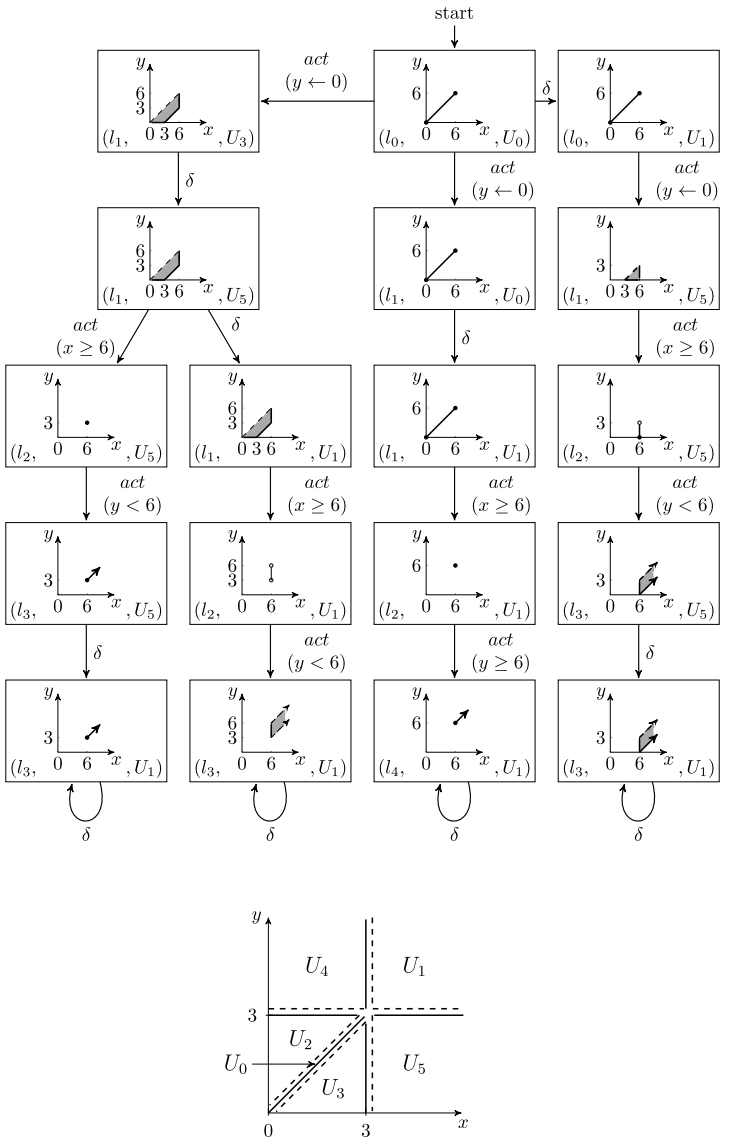


Fig. 5. ZURA state space of timed automaton $A_{4.1}$ with respect to $G = \{x \leq 3, y \leq 3\}$.

- $D'_{ij} = x_i - x_j < \infty$ and $C'_{ij} = \{e_{ij} > M(x_i)\}$,
- $D'_{ij} = x_i - x_j \prec_{ij} e_{ij}$ and $C'_{ij} = \{-M(x_j) \leq e_{ij}, e_{ij} \leq M(x_i)\}$,
- $D'_{ij} = x_i - x_j < -M(x_j)$ and $C'_{ij} = \{e_{ij} < -M(x_j)\}$.

In the rest of the paper we use $[U]_c$ to denote the operation $[g_U]_c$ where U is an ultraregion and g_U is a conjunction of simple guards defining the ultraregion U. We now continue with a definition of the abstract symbolic semantics of a parametric timed automaton based on CPDBMs and ultraregions using the pk-extrapolation abstraction.

Definition 5.2 *(PTA abstract symbolic semantics with ultraregions).* Let $A = (L, l_0, X, P, \Delta, Inv)$ be a PTA. Let lb be a lower bound function and ub be an upper bound function on parameters P. Let $G \subseteq \overline{G}_1(X, \emptyset)$ be a set of simple non-diagonal guards without parameters. The *symbolic semantics* of A with respect to lb, ub and G is a labelled transition system $(\mathbb{S}_A, \mathbb{S}_{init}, \Longrightarrow)$ over a set *Label*, denoted as $[\![A]\!]^G_{lb,ub}$, where

- $Label = \{act, \delta\}$,
- $\mathbb{S}_A = L \times \{[\![C, D]\!] \mid (C, D)$ is a $CPDBM\} \times \{U \mid U$ is an ultraregion of $G\}$ is the set of all symbolic states,
- the set of initial states $\mathbb{S}_{init} = \{(l_0, [\![C, D]\!], U) \mid (C, D) \in (\hat{C}, E^{\uparrow})[Inv(l_0)]\}$, where

- E is a PDBM with $E_{i,j} = (0, \leq)$ for each i, j, and
- $\hat{C} = \{p \geq lb(p) \mid p \in P\} \cup \{p \leq ub(p) \mid p \in P\}$, and
- U_0 is the ultraregion containing the zero valuation $\mathbf{0}$.

- the transitions are of two kinds:
  - an action transition $(l, [\![C, D]\!], U) \xRightarrow{act} (l', [\![C', D']\!], U')$ whenever
    - $l \xrightarrow{g,R}_{\Delta} l'$ and
    - $U' \in U\lfloor R \rfloor$ and
    - there exist satisfiable CPDBMs $(C_1, D_1), (C_2, D_2), (C_3, D_3), (C_4, D_4)$ such that:
      - $(C_1, D_1) \in (C, D)[U]_c[g]_c$ and
      - $(C_2, D_2) \in (C_1, D_1\langle R \rangle)[U']_c$ and
      - $(C_3, D_3) \in (C_2, D_2^{\uparrow})[Inv(l')]_c$ and
      - $(C_4, D_4) \in (C_3, D_3)[U']_c$ and $[\![C_4, D_4]\!] \neq \emptyset$ and
      - $(l', [\![C', D']\!], U') \in \alpha_{pk}(l', [\![C_4, D_3]\!], U')$.
  - a delay transition $(l, [\![C, D]\!], U) \xRightarrow{\delta} (l, [\![C', D]\!], succ(U))$ whenever
    - $(C', D') \in (C, D)[succ(U)]_c$ and $[\![C', D']\!] \neq \emptyset$ and
    - for each $v \in [\![C']\!]$ holds $U = succ(U) \Rightarrow [\![D]\!]_v = [\![D^{\uparrow}]\!]_v$ and
    - $[\![C', D]\!] \neq \emptyset$.

We say that $\pi = S_0 \xRightarrow{\gamma_0} S_1 \xRightarrow{\gamma_1} \dots$ is a run of $[\![A]\!]^G_{lb,ub}$ if $S_0 \in \mathbb{S}_{init}$ and for each $i \in \mathbb{N}_{>0}$, $S_i \in \mathbb{S}_A$ and $S_{i-1} \xRightarrow{\gamma_{i-1}} S_i$. A run is called a v-run if for each state $S_i = (l_i, [\![C_i, D_i]\!], U_i)$ it holds that $v \in [\![C_i]\!]$. We say that state S is v-reachable if there exists a v-run $\pi = S_0 \xRightarrow{\gamma_0} \dots \xRightarrow{\gamma} S$. We use A_{PZURA} to denote $[\![A]\!]^G_{lb,ub}$ if G, lb, ub is clear from the context.

Example 5.3. We illustrate the PTA abstract symbolic semantics with ultraregions on the sensor PTA from Example 2.7 with respect to $G = \{x \leq 3\}$. The corresponding ultraregions are $U_0 = \{(x, y) | x \in [0, 3], y \in [0, \infty)\}$ and $U_1 = \{(x, y) | x \in (3, \infty), y \in [0, \infty)\}$. Consider $X = \{x, y\}$, $P = \{p, q\}$, $M(x) = M(y) = 5$ and the initial parameter bounds $3 \leq p \leq 4, 2 \leq q \leq 4$. We perform several transitions in the abstract symbolic semantics with ultraregions.

The initial state is $S_0 = (l_0, [\![C_0, D_0]\!], U_0) = (l_0, [\![\{3 \leq p \leq 4, 2 \leq q \leq 4\}, \{0 \leq x < \infty, 0 \leq y < \infty, 0 \leq y - x \leq 0\}]\!], U_0)$. First, we execute the delay transition in S_0 and obtain $S_1 = (l_0, [\![C_0, D_0]\!], U_1)$. When the execution of the PTA transition from l_0 to l_1 is considered in the state S_1, we obtain one successor $S_2 = (l_1, [\![C_0, D_2]\!], U_0) = (l_1, [\![C_0, \{0 \leq x \leq p, 0 \leq y \leq p, 0 \leq y - x \leq 0\}]\!], U_0)$. When the delay transition is executed in S_2, the state $S_3 = (l_1, [\![C_0, D_2]\!], U_1)$ is obtained. When the execution of the PTA loop transition from l_1 to l_1 is considered in the state S_2, we obtain two successors $S_4 = (l_1, [\![C_0 \cup \{q \leq p, p \leq 3\}, \{0 \leq x \leq p, q \leq y < \infty, q \leq y - x \leq p\}]\!], U_0)$ and $S_5 = (l_1, [\![C_0 \cup \{q \leq 3, 3 < p\}, \{0 \leq x \leq p, q \leq y < \infty, q \leq y - x \leq 3\}]\!], U_0)$.

Analogously to Lemma 3.14 the PTA abstract symbolic semantics with ultraregions has the property of monotonicity which means that the set of parameter valuations associated with each state can not grow along any run. Note the fact that PTA abstract symbolic semantics with ultraregions can be viewed as an LTS with a monotonic annotation f such that $f(l, [\![C, D]\!], U) = [\![C]\!]$ for each state $(l, [\![C, D]\!], U)$ of this semantics.

Theorem 5.4. *Let A be a PTA. Let lb be a lower bound function and ub be an upper bound function on parameters P. Let $G \subseteq \overline{G}_1(X, \emptyset)$ be a set of simple non-diagonal guards without parameters. The LTS $[\![A]\!]^G_{lb,ub}$ is finite.*

Proof. Note the fact that set of all ultraregions for a given finite G is always finite. Therefore this theorem is just corollary to Theorem 3.15. □

We now state the main result of the section which allows us to use standard LTL model checking technique in the solution of our problem. That means we are able to construct a BA with a monotonic annotation from an LTS A_{PZURA} with a monotonic annotation f and LTL formula φ.

Theorem 5.5. *Let A be a PTA. Let lb be a lower bound function and ub be an upper bound function on parameters P. Let $G \subseteq \overline{G}_1(X, \emptyset)$ be a set of simple non-diagonal guards without parameters. Let $\mathcal{L}$ be a location labelling function. Let further φ be a CA-LTL formula over G. Then for each parameter valuation v holds $([\![A|_v]\!], \mathcal{L}) \models \varphi$ iff $([\![A]\!]^G_{lb,ub}, v, \mathcal{L}) \models \varphi$.*

We continue with a remark about Zeno runs. It might sometimes happen that the synthesis algorithm rejects a parameter valuation v such that the produced counterexample is a Zeno v-run of the original PTA. If ignoring such runs is desirable (as it usually is), we may extend the original PTA with one special clock z, add a loop on every location with guard $z = 1$ and reset $\{z\}$, and modify the original CA-LTL formula from φ to $(\mathbf{GF}z \leq 0 \wedge \mathbf{GF}z > 0) \Rightarrow \varphi$. Note the fact that the standard transformation to the strongly non-Zeno form [30] is applicable to PTBA only and thus cannot be employed here.

5.2. Proof of Theorem 5.5

Let us assume a fixed PTA $A = (L, l_0, X, P, \Delta, Inv)$, a fixed lb, ub functions, a fixed set of simple non-diagonal guards without parameters $G \subseteq \overline{G}_1(X, \emptyset)$, and a fixed location labelling function $\mathcal{L}$.

We show that for each parameter valuation v the proper runs of $[\![A|_v]\!]$ and the v-runs of A_{PZURA} are, in some sense, equivalent. In order to do that, we use the notion of a signature of a run. Intuitively, a signature is a sequence of sets of atomic propositions that hold along the given run.

We use the fact that all clock valuations of a given ultraregion satisfy the same set of guards. For an ultraregion U, we thus use $G(U)$ to denote the set of guards satisfied by the valuations of U.

Definition 5.6 *(Signature).* Let $G \subseteq \overline{G}_1(X, \emptyset)$ be a set of simple non-diagonal guards without parameters, Lp a set of location propositions, $Ap = G \cup Lp$, and let $\pi = (l_0, \eta_0) \xrightarrow{d_0} (l_0, \eta_0 + d_0) \xrightarrow{act} (l_1, \eta_1) \cdots$ be a proper run of a PTA. Let $U_{j,0}$ be the ultraregion of G containing η_j and let $U_{j,i+1} = succ(U_{j,i})$. For $(l_j, \eta_j) \xrightarrow{d_j} (l_j, \eta_j + d_j)$ in π, we define $w_j \in (2^{Ap})^+$:

$$w_j = (\mathcal{L}(l_j) \cup G(U_{j,0})) \cdot (\mathcal{L}(l_j) \cup G(U_{j,1})) \cdots (\mathcal{L}(l_j) \cup G(U_{j,k}))$$

where k is the least such that $U_{j,k} = U_{j,k+1}$. For $(l_j, \eta_j) \xrightarrow{\infty} (l_j, \infty)$, we define:

$$w_j = (\mathcal{L}(l_j) \cup G(U_{j,0})) \cdot (\mathcal{L}(l_j) \cup G(U_{j,1})) \cdots$$

In this case, $w_j \in (2^{Ap})^\omega$. The signature of π with respect to Ap, denoted by $sig_{Ap}(\pi)$, is defined as the infinite word $w = w_0 w_1 w_2 \cdots w_j$ if π ends with (l_j, ∞), $w = w_0 w_1 w_2 \cdots$ otherwise.

Our first objective is to show that for each parameter valuation v the proper runs of $[\![A|_v]\!]$ and the v-runs of A_{PZURA} are equivalent in the sense that a signature of a proper run of $[\![A|_v]\!]$ is the sequence of labellings on a v-run of A_{PZURA} and vice versa. Detailed proofs of Lemma 5.7 and Lemma 5.8 can be found in Appendix B.

Lemma 5.7. *Let v be a parameter valuation and let π be a proper run of $[\![A|_v]\!]$ with signature $sig_{Ap}(\pi)$. Then there exists an infinite v-run of A_{PZURA} $(l_0, [\![C_0, D_0]\!], U_0) \xRightarrow{\gamma_0} (l_1, [\![C_1, D_1]\!], U_1) \xRightarrow{\gamma_1} \cdots$ with $\gamma_i \in \{act, \delta\}$ such that $sig_{Ap}(\pi)(i) = \mathcal{L}(l_i) \cup G(U_i)$, where $sig_{Ap}(\pi)(i)$ denotes the ith letter (from 2^{Ap}) of $sig_{Ap}(\pi)$.*

Lemma 5.8. *Let v be a parameter valuation and let $(l_0, [\![C_0, D_0]\!], U_0) \xRightarrow{\gamma_0} (l_1, [\![C_1, D_1]\!], U_1) \xRightarrow{\gamma_1} \cdots$ be a v-run of A_{PZURA} with $\gamma_i \in \{act, \delta\}$. Then there exists a proper run π of $[\![A|_v]\!]$ such that $sig_{Ap}(\pi)(i) = \mathcal{L}(l_i) \cup G(U_i)$.*

What remains is to show that the CA-LTL satisfaction on a proper run can be reduced to the classic LTL satisfaction on its signature.

Let us recall that the classic satisfaction relation for LTL on an infinite word from $(2^{Ap})^\omega$ is given by $w \models p \in Ap$ if $p \in w(0)$, $w \models \varphi \,\mathbf{U}\, \psi \iff \exists k : w^k \models \psi$ and for all $j < k : w^j \models \varphi$. Here, $w(0)$ is the first letter of the word w and w^k is the kth suffix of w. We omit the standard Boolean operators.

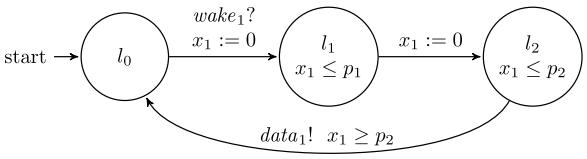
Lemma 5.9. *[12] Let π be a proper run of a TA, let G be a set of simple non-diagonal guards, Lp a set of location propositions, $Ap = G \cup Lp$. Let φ be a CA-LTL formula over Ap. Then $\pi \models \varphi$ iff $sig_{Ap}(\pi) \models \varphi$ when seen as an LTL formula.*

These results together imply the statement of Theorem 5.5.

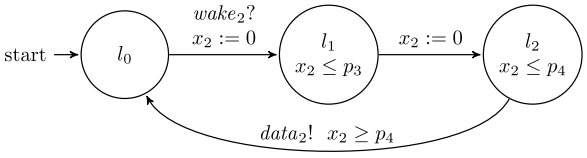
5.3. Parameter synthesis algorithm

In the Subsection 5.1 we have constructed a BA with a monotonic annotation as a result of the second step of our algorithmic framework for CA-LTL parameter synthesis. We now employ the Cumulative NDFS algorithm introduced in the Subsection 3.2. To conclude, we state that Theorem 5.5 together with Theorem 5.4 and Theorem 3.16 imply the correctness of our solution to Problem 2.9.

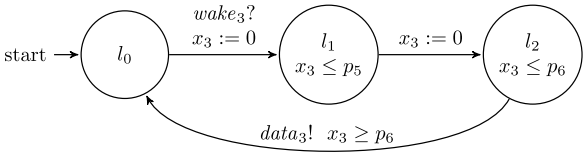
It is also possible to apply the state space storage technique based on integer hulls discussed in Subsection 3.4. As we are working with a BA obtained from abstract symbolic semantics with ultraregions of PTA, we are allowed to reference the l, C, D, U components of each state. The only technicality to solve is the trivial addition of the fourth component to a symbolic state stored in the state space storage. The addition of the fourth ultraregion component of a symbolic state does not affect efficiency of the state space storage operations introduced in Subsection 3.4. The ultraregion implementation details are described in [12].



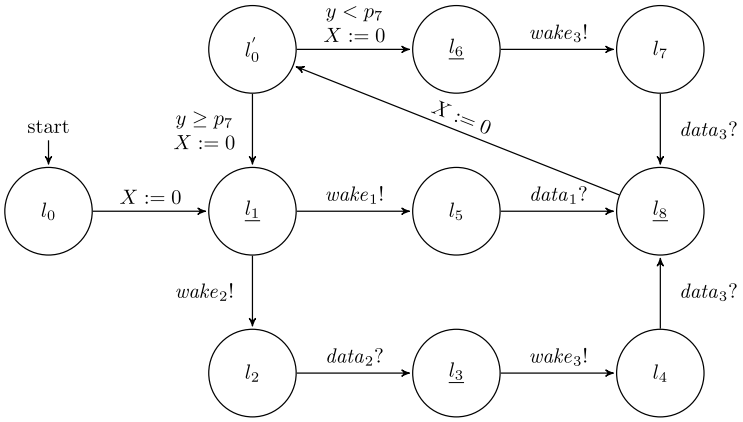
(a) Sensor 1 with parameters p_1 and p_2



(b) Sensor 2 with parameter p_3 and p_4



(c) Sensor 3 with parameter p_5 and p_6



(d) Controller with parameter p_7

Fig. 6. Sensors subsystem.

5.4. Experimental evaluation

We have implemented the proposed technique for bounded integer parameter synthesis in our proof-of-concept tool[2] [33]. Our goal is to compare the CNDFS algorithm with the explicit enumeration technique. To be able to compare the performance of both techniques under similar conditions we have also implemented the standard DBM-based CA-LTL model checker for timed automata. Both tools use analogous extrapolation techniques. Checking whether two states are semantically equivalent is implemented using the Parma Polyhedra Library [34]. The library is also used to check parametric constraint satisfaction in the CPDBM operations. We used the LTL to BA translation method from [35] and the integer hull computation method from [11].

We have performed the evaluation on a sensors subsystem model which is presented in Fig. 6. The sensors subsystem is modelled using a parametric timed network which is a standard extension of parametric timed automata. The parametric timed network consists of several parametric timed automata where each parametric timed automaton can perform its transitions independently. It is also allowed to perform a synchronisation of two components using transitions labelled with

[2] Available online at https://paradise.fi.muni.cz/parameterSynthesis/index.html.

a matching channel (e.g. $wake_1!$, $wake_1?$). The underlined locations denote the so-called urgent locations where time is not allowed to pass.

The sensors subsystem is responsible for the gathering of data from several sensors and providing one final value derived from the gathered data. This final value is used later when the sensors subsystem is combined with other components of a system. The model consists of a controller and 3 sensors. The controller is responsible for waking up the appropriate sensors. There is a time limit, represented by p_7, in which the previously gathered data can be reused and it is sufficient to gather data from the sensor 3 only. Note the fact that $p_7 = 0$ effectively disables the data reuse. After p_7 time units the controller needs to gather data either from both of the sensor 2 and 3 or from the sensor 1 only. This nondeterministic choice represents different options to derive the final value. The first option combines data gathered from two sensors, while the second option requires data only from one sensor which is usually more accurate but slower. The controller location l_0 represents an initial idle state and the location l'_0 represents a general idle state. Each sensor consists of 3 locations. The location l_0 represents the idle state, the location l_1 represents a sensor initialisation phase with a variable duration between 0 and some upper time limit (e.g. p_1 in case of the sensor 1), and the location l_2 represents the data gathering phase with a fixed duration (e.g. p_2 in case of the sensor 1).

For the purpose of the evaluation we consider the following 3 properties:

- $\varphi_1 = \mathbf{G}((Controller.l_1 \vee Controller.l_6) \Rightarrow (y \leq 500\, \mathbf{U}\, Controller.l_8))$
- $\varphi_2 = \mathbf{G}((Controller.l_1 \vee Controller.l_6) \Rightarrow (y \leq 150\, \mathbf{U}\, Controller.l_8))$
- $\varphi_3 = \mathbf{G}((Controller.l_1 \vee Controller.l_6) \Rightarrow \mathbf{F}\, Controller.l_8)$

The first property φ_1 states that whenever the controller starts the measurement it will finish the measurement eventually and the clock y will not exceed 500 time units until such finish. The second property φ_2 states that whenever the controller starts the measurement it will finish the measurement eventually and the clock y will not exceed 150 time units until such finish. The third property φ_3 states that whenever the controller starts the measurement it will finish the measurement eventually. The properties φ_1 and φ_3 are satisfied for all considered parameter valuations, whereas the satisfaction of φ_2 depends on a particular parameter valuation. The property φ_2 is not satisfied whenever it holds that $p_1 + p_2 > 150$ or $p_3 + p_4 + p_5 + p_6 > 150$.

The experiments were performed on a PC with CPU i5-4690 and 16 GB RAM. We consider a timeout (TO) of 2 hours for each task. We provide a percentage of solved parameter valuations if the timeout was reached by the explicit enumeration. In case of the explicit enumeration, the provided time is a sum of running times of all individual model checking instances and the resulting time excludes the initialisation overhead of each model checking instance.

Table 1 shows the impact of the number of parameters used in the model. We evaluate the explicit enumeration technique and the CNDFS algorithm for each combination of a property and a parameter count. Moreover, we use φ_2^* and φ_3^* to denote additional runs of the CNDFS algorithm where a larger maximum constant 500 is considered in the pk-extrapolation. We have added these two rows to show the counter intuitive fact that a lower running time can be reached with a maximum constant that is larger than necessary. The reason is the fact that a larger maximum constant can lead to less frequent CPDBM splits during the application of the pk-extrapolation operation, and consequently the reachable state space can be smaller. Table 1 clearly shows that higher parameter count significantly favours the CNDFS algorithm.

Table 2 shows the impact of the parameter range size on the execution times. As expected, the execution time of the explicit enumeration depends on the parameter range size. On the other hand, the execution time of cumulative algorithm is not affected by the parameter range size directly. For example, the φ_3 CNDFS execution time for a range $[51, 100]$ is significantly lower than the time for a range $[1, 50]$ despite the same parameter range size. In contrast, the φ_3 CNDFS execution time for a range $[1, 50]$ is similar to the time for a range $[1, 10]$ despite a different parameter range size. The execution time of the CNDFS algorithm depends on a particular combination of the property and the parameter range. For example, the CNDFS algorithm is slower for φ_2 when the considered parameter range contains values larger than 50 (e.g. $[51, 100]$ or $[1, 100]$). On the other hand, the CNDFS algorithm is slower for φ_1 when the considered parameter range contains small values (e.g. $[1, 10]$, $[1, 50]$ or $[1, 100]$). This shows that the exact performance of CNDFS may be hard to predict in general.

6. Conclusion and future work

We have introduced a symbolic method which solves the parameter synthesis problem for Clock-Aware LTL properties and timed automata with bounded integer parameters. The novelty of this publication is based on the introduction of the new symbolic abstraction and corresponding abstract semantics. This new symbolic abstraction combines the abstraction proposed in the SEFM 2016 conference paper [23] with the ideas of Clock-Aware LTL model checking described in [12].

The proposed symbolic technique allows the avoidance of the explicit enumeration of all possible parameter valuations. The symbolic technique is based on the zone abstraction and uses a parametric extension of difference bound matrices. To employ the zone-based method successfully we need to apply a finite abstraction called the pk-extrapolation. To be able to synthesise all violating parameter valuations we have used the Cumulative NDFS algorithm which is an extension of the NDFS algorithm. In order to support the Clock-Aware LTL specification we have applied the ultraregions technique [12] when

Table 1
Impact of model parameter count.

Sensors subsystem model	2 params	3 params	4 params	5 params	6 params	7 params
	$p_1 \in [1, 100]$	$p_1 \in [1, 100]$	$p_1 \in [1, 100]$	$p_1 \in [1, 100]$	$p_1 \in [1, 100]$	$p_1 \in [1, 100]$
	$p_2 \in [1, 100]$	$p_2 \in [1, 100]$	$p_2 \in [1, 100]$	$p_2 \in [1, 100]$	$p_2 \in [1, 100]$	$p_2 \in [1, 100]$
	$p_3 = 5$	$p_3 \in [1, 100]$	$p_3 \in [1, 100]$	$p_3 \in [1, 100]$	$p_3 \in [1, 100]$	$p_3 \in [1, 100]$
	$p_4 = 100$	$p_4 = 100$	$p_4 \in [1, 100]$	$p_4 \in [1, 100]$	$p_4 \in [1, 100]$	$p_4 \in [1, 100]$
	$p_5 = 1$	$p_5 = 1$	$p_5 = 1$	$p_5 \in [1, 100]$	$p_5 \in [1, 100]$	$p_5 \in [1, 100]$
	$p_6 = 50$	$p_6 = 50$	$p_6 = 50$	$p_6 = 50$	$p_6 \in [1, 100]$	$p_6 \in [1, 100]$
	$p_7 = 0$	$p_7 = 0$	$p_7 = 0$	$p_7 = 0$	$p_7 = 0$	$p_7 \in [0, 100]$
φ_1 explicit	3.5 s	351 s	TO (17%)	TO (0%)	TO (0%)	TO (0%)
φ_1 CNDFS	0.4 s	2.2 s	3.3 s	5.7 s	8.6 s	36 s
φ_2 explicit	2.5 s	302 s	TO (20%)	TO (0%)	TO (0%)	TO (0%)
φ_2 CNDFS	2 s	25 s	151 s	1188 s	4924 s	TO
φ_2^* CNDFS	2.5 s	29 s	193 s	866 s	3120 s	TO
φ_3 explicit	1.7 s	213 s	TO (22%)	TO (0%)	TO (0%)	TO (0%)
φ_3 CNDFS	0.5 s	3.9 s	52 s	124 s	189 s	1383 s
φ_3^* CNDFS	0.3 s	1.5 s	2 s	3.8 s	5.6 s	24 s

Table 2
Impact of parameter range size.

Sensors subsystem model	$p_1 \in [1, 10]$	$p_1 \in [1, 50]$	$p_1 \in [51, 100]$	$p_1 \in [1, 100]$
6 parameters and $p_7 = 0$	$p_2 \in [1, 10]$	$p_2 \in [1, 50]$	$p_2 \in [51, 100]$	$p_2 \in [1, 100]$
	$p_3 \in [1, 10]$	$p_3 \in [1, 50]$	$p_3 \in [51, 100]$	$p_3 \in [1, 100]$
	$p_4 \in [1, 10]$	$p_4 \in [1, 50]$	$p_4 \in [51, 100]$	$p_4 \in [1, 100]$
	$p_5 \in [1, 10]$	$p_5 \in [1, 50]$	$p_5 \in [51, 100]$	$p_5 \in [1, 100]$
	$p_6 \in [1, 10]$	$p_6 \in [1, 50]$	$p_6 \in [51, 100]$	$p_6 \in [1, 100]$
φ_1 explicit	427 s	TO (0%)	TO (0%)	TO (0%)
φ_1 CNDFS	8.4 s	8.4 s	8.5 s	8.6 s
φ_2 explicit	426 s	TO (0%)	TO (0%)	TO (0%)
φ_2 CNDFS	8.4 s	33 s	1231 s	4924 s
φ_2^* CNDFS	8.4 s	35 s	864 s	3120 s
φ_3 explicit	357 s	TO (0%)	TO (0%)	TO (0%)
φ_3 CNDFS	189 s	190 s	6.6 s	189 s
φ_3^* CNDFS	6.2 s	6.2 s	6.2 s	6.3 s

building the symbolic state space. We have implemented the parameter synthesis algorithm in an experimental tool. Our experiments confirm that the proposed algorithm can be significantly faster than the explicit enumeration technique.

As for future work we plan to introduce different finite abstractions based on different extrapolations and compare their influence on the state space size. We also plan to introduce a parallel version of the cumulative algorithm. Another area that can be investigated is the employment of parameters in specification logics, e.g. parametric extension of Clock-Aware LTL which enables the use of parameters in clock constraints.

Appendix A. Proof of Theorem 3.15

A.1. Finiteness of pk-extrapolation

Lemma A.1. *Let A be a PTBA. The pk-extrapolation is a finite abstraction over $[\![A]\!] = (\mathbb{S}_A, \mathbb{S}_{init}, \Longrightarrow)$.*

Proof. We first prove that the pk-extrapolation is an abstraction. It is easy to see that the pk-extrapolation satisfies the first condition $(l', [\![C', D']\!]) \in \alpha((l, [\![C, D]\!]))$ implies $l = l' \wedge [\![C']\!] \subseteq [\![C]\!] \wedge [\![C', D]\!] \subseteq [\![C', D']\!]$. The validity of the second condition follows from the following observation. For each $v \in [\![C]\!]$ and each $\eta' \in [\![D']\!]_v$ there exists $\eta \in [\![D]\!]_v$ such that for each clock x and each guard g the following implication holds: $\eta'(x) \models g \implies \eta(x) \models g$.

We now need to show that the pk-extrapolation is finite. From the definition we have the fact that the number of locations is finite and the number of sets of bounded parameter valuations is finite. We need to show that there are only finitely many sets $[\![C, D]\!]$ when the pk-extrapolation is applied. This follows from the fact that for each e_{ij} from D and $v \in [\![C]\!]$ the expression $[\![e_{ij}]\!]_v$ can be evaluated only to a value from the finite set $\{-M(x_i), -M(x_i) + 1, \ldots, M(x_i) - 1, M(x_i), \infty\}$. □

A.2. Preservation of accepting runs

We transform the proof of Theorem 1 of [36] and all corresponding lemmata into our parametric setup. For the sake of simplicity of the proof, we add labels to the transitions in $[\![A]\!]^\alpha$ in the following way. For each transition we use the location of a source state as the transition label. Since labels are not used in the proposed method, it is safe to do that.

For the rest, let v be a parameter valuation, A be a PTBA, and α be a finite abstraction over $[\![A]\!]$. Then, we denote by $A \mid v$ a timed Büchi automaton obtained from A by replacing each parameter p with the value $v(p)$. We use $\equiv_{A|v}$ to denote the standard region abstraction [36] over the timed Büchi automaton $A \mid v$.

We write $s_1 \xrightarrow{act_1,act_2,\ldots,act_{k-1}}_d s_k$ if there exist $s_2, \ldots, s_{k-1}$ such that $s_1 \xrightarrow{act_1}_d s_2$, $s_2 \xrightarrow{act_2}_d s_3$, ..., and $s_{k-1} \xrightarrow{act_{k-1}}_d s_k$. We write $s \xrightarrow{act_1,act_2,\ldots,act_k}{}^*_d s'$ if $s \xrightarrow{act_1,act_2,\ldots,act_k}_d s'$ or there exist some $s_1, s_2, \ldots, s_n (n \geq 1)$ such that $s \xrightarrow{act_1,act_2,\ldots,act_k}_d s_1, s_1 \xrightarrow{act_1,act_2,\ldots,act_k}_d s_2, \ldots, s_{n-1} \xrightarrow{act_1,act_2,\ldots,act_k}_d s_n$, and $s_n \xrightarrow{act_1,act_2,\ldots,act_k}_d s'$.

Lemma A.2. *[36] The equivalence relation $\equiv_{A|v}$ is a time-abstracting bisimulation.*

Lemma A.3. *[36] Let s_1, s_1', s_2 be concrete states in $[\![A]\!]_v$, R be a time-abstracting simulation and $s_1 R s_1'$. If $s_1 \xrightarrow{act_1,act_2\ldots,act_k}_d s_2$, then there exists a concrete state s_2' in $[\![A]\!]_v$ such that $s_1' \xrightarrow{act_1,act_2\ldots,act_k}_d s_2'$.*

Lemma A.4. *Let s_1, s_2 be concrete states in $[\![A]\!]_v$. If $s_1 \xrightarrow{act_1,act_2\ldots,act_k}_d s_2$ and $s_1 \equiv_{A|v} s_2$, then there is an infinite sequence of concrete states $s_1 s_2 \ldots$ in $[\![A]\!]_v$ such that for each $i \geq 1$, $s_i \xrightarrow{act_1,act_2,\ldots,act_k}_d s_{i+1}$.*

Proof. We define s_k $(k = 3, 4, \ldots)$ by induction on k.

Basis: By Lemma A.2 and Lemma A.3 there exists s_3 such that $s_2 \equiv_{A|v} s_3$ and $s_2 \xrightarrow{act_1,act_2,\ldots,act_k}_d s_3$.

Assumption: Assume that we have $s_1, s_2, \ldots, s_k$ such that for each $i \in \{1, 2, \ldots, k-1\}$, $s_i \xrightarrow{act_1,act_2,\ldots,act_k}_d s_{i+1}$, and $s_{k-1} \equiv_{A|v} s_k$.

Step: From $s_{k-1} \xrightarrow{act_1,act_2,\ldots,act_k}_d s_k$, and $s_{k-1} \equiv_{A|v} s_k$, by Lemma A.3 there exists s_{k+1} such that $s_k \xrightarrow{act_1,act_2,\ldots,act_k}_d s_{k+1}$, and $s_k \equiv_{A|v} s_{k+1}$.

Thus, using induction, we get an infinite sequence of states $s_1 s_2 \ldots$ such that for each $i \geq 1$, $s_i \xrightarrow{act_1,act_2,\ldots,act_k}_d s_{i+1}$. □

Lemma A.5. *Let s', s_1, s_2 be concrete states in $[\![A]\!]_v$ and S_1, S_2 be symbolic states in $[\![A]\!]$.*

1. *If $S_1 \Longrightarrow S_2$ and $s' \in_v S_2$, then there exist concrete state s in $[\![A]\!]_v$ such that $s \xrightarrow{act}_d s'$.*
2. *If $s_1 \xrightarrow{act}_d s_2$ and $s_1 \in_v S_1$, then $S_1 \Longrightarrow S_2$ for some symbolic state S_2 in $[\![A]\!]$ with $s_2 \in_v S_2$.*

Proof. We refer the reader to the proofs of Lemma 3.16 and Lemma 3.18 in [9]. □

Lemma A.6. *Let s, s_1, s_2 be concrete states in $[\![A]\!]_v$, and Q_1, Q_2 be symbolic states in $[\![A]\!]^\alpha$.*

1. *If $Q_1 \Longrightarrow_\alpha Q_2$ and $s \in_v Q_2$, then there exist concrete states s_1', s_2' in $[\![A]\!]_v$ such that $s_1' \xrightarrow{act}_d s_2'$, $s_1' \in_v Q_1$ and $s \preccurlyeq s_2'$.*
2. *If $s_1 \xrightarrow{act}_d s_2$ and $s_1 \in_v Q_1$, then $Q_1 \Longrightarrow_\alpha Q_2$ for some symbolic state Q_2 in $[\![A]\!]^\alpha$ with $s_2 \in_v Q_2$.*

Proof.

1. From $Q_1 \Longrightarrow_\alpha Q_2$ we know that there exists S such that $Q_1 \Longrightarrow S$ and $Q_2 \in \alpha(S)$. For any $s \in_v Q_2$, since $Q_2 \in \alpha(S)$, there is $s_2' \in_v S$ such that $s \preccurlyeq s_2'$. Since $Q_1 \Longrightarrow S$ and $s_1' \in_v S$, by Lemma A.5, there is a $s_1' \in_v Q_1$ such that $s_1' \xrightarrow{act}_d s_2'$.
2. By Lemma A.5 there is a S such that $Q_1 \Longrightarrow S$ with $s_2 \in_v S$. Let $Q_2 \in \alpha(S)$ then $Q_1 \Longrightarrow_\alpha Q_2$ and $s_2 \in_v Q_2$. □

Lemma A.7. *Let s be a concrete state in $[\![A]\!]_v$, Q_1, Q_2 be symbolic states in $[\![A]\!]^\alpha$. If $Q_1 \xRightarrow{act_1,act_2,\ldots,act_k} Q_2$, and $s \in_v Q_2$, then there exist concrete states s_1, s_2 in $[\![A]\!]_v$ such that $s_1 \in_v Q_1$, $s_1 \xrightarrow{act_1,act_2,\ldots,act_k}_d s_2$, and $s \preccurlyeq s_2$.*

Proof. We prove the lemma by induction on k.

Basis: By Lemma A.6, the lemma is true for $k = 1$.

Assumption: Assume that lemma holds for $k = n$.

Step: Now we prove the lemma for $k = n + 1$. $Q_1 \xRightarrow{act_1,act_2,\ldots,act_{n+1}}_\alpha Q_2$ implies that there exists a $Q \in [\![A]\!]^\alpha$ such that $Q_1 \xRightarrow{act_1,act_2,\ldots,act_n}_\alpha Q$ and $Q \xRightarrow{act_{n+1}}_\alpha Q_2$. By Lemma A.6 and the fact that $Q \xRightarrow{act_{n+1}}_\alpha Q_2$ and $s \in_v Q_2$, we have s' and s'' such that $s' \in_v Q$, $s' \xrightarrow{act_{n+1}}_d s''$, and $s \preccurlyeq s''$. Since $Q_1 \xRightarrow{act_1,act_2,\ldots,act_n}_\alpha Q$ and $s' \in_v Q$, by the induction assumption there exist s_1 and s''' such that $s_1 \in_v Q_1$, $s_1 \xrightarrow{act_1,act_2,\ldots,act_n}_d s'''$,and $s' \preccurlyeq s'''$. Since $s' \preccurlyeq s'''$ and $s' \xrightarrow{act_{n+1}}_d s''$, by Lemma A.3 it

follows that there is a s_2 such that $s''' \xrightarrow{act_{n+1}}_d s_2$ and $s'' \preccurlyeq s_2$. From the fact that $\preccurlyeq$ is transitive and $s \preccurlyeq s''$ and $s'' \preccurlyeq s_2$ we have $s \preccurlyeq s_2$.

By $s_1 \xrightarrow{act_1, act_2, \ldots, act_n}_d s'''$ and $s''' \xrightarrow{act_{n+1}}_d s_2$ we obtain $s_1 \xrightarrow{act_1, act_2, \ldots, act_n+1}_d s_2$. □

Lemma A.8. *Let s be a concrete state in $[\![A]\!]_v$, Q_1, Q_2 be symbolic states in $[\![A]\!]^\alpha$. If $Q \xRightarrow{act_1, act_2, \ldots, act_k}_\alpha Q$ and $s \in_v Q$, then for any $n \geq 1$, there exist concrete states $s_1, s_2 \ldots s_{n+1}$ in $[\![A]\!]_v$ such that $s_1 \in_v Q$, $s \preccurlyeq s_{n+1}$, and for each $i \in 1, 2, \ldots, n$, $s_i \xrightarrow{act_1, act_2, \ldots, act_k}_d s_{i+1}$.*

Proof. We prove the lemma by induction on n.

Basis: By Lemma A.7, the lemma is true for $n = 1$.

Assumption: Assume that lemma holds for $n = m$.

Step: We now prove that the lemma is true for $n = m + 1$.

Since $Q \xRightarrow{act_1, act_2, \ldots, act_k}_\alpha Q$ and $s \in_v Q$, by Lemma A.7, there exist s', s'' such that $s' \in_v Q$, $s' \xrightarrow{act_1, act_2, \ldots, act_k}_d s''$, and $s \preccurlyeq s''$. Applying the induction assumption to $Q \xRightarrow{act_1, act_2, \ldots, act_k}_\alpha Q$ and $s' \in_v Q$, we know that there exist $s_1, s_2 \ldots s_{m+1}$ such that $s_1 \in_v Q$, $s' \preccurlyeq s_{m+1}$, and for each $i \in 1, 2, \ldots, m$, $s_i \xrightarrow{act_1, act_2, \ldots, act_k}_d s_{i+1}$.

Since $s' \xrightarrow{act_1, act_2, \ldots, act_k}_d s''$ and $s' \preccurlyeq s_{m+1}$, by Lemma A.3, there exists s_{m+2} such that $s_{m+1} \xrightarrow{act_1, act_2, \ldots, act_k}_d s_{m+2}$, and $s'' \preccurlyeq s_{m+2}$.

Since $s \preccurlyeq s''$ and $s'' \preccurlyeq s_{m+2}$ we obtain $s \preccurlyeq s_{m+2}$, thus the lemma holds for $n = m + 1$. □

Lemma A.9. *Let s be a concrete state in $[\![A]\!]_v$, Q_1, Q_2 be symbolic states in $[\![A]\!]^\alpha$. If $Q \xRightarrow{act_1, act_2, \ldots, act_k}_\alpha Q$ and $s \in_v Q$, then there exist concrete states $s_1, s_2 \ldots s_m$ in $[\![A]\!]_v$ and $i \in \{1, 2, \ldots, m-1\}$ such that $s_1 \in_v Q$, $s_i \equiv_{A|v} s_m$, and for each $j \in \{1, 2, \ldots, m-1\}$, $s_j \xrightarrow{act_1, act_2, \ldots, act_k}_d s_{j+1}$.*

Proof. We know that there are only finitely many $\equiv_{A|v}$-equivalence classes. Let n be an integer greater than the number of $\equiv_{A|v}$-equivalence classes. By Lemma A.8, there exist $s_1, s_2 \ldots s_{n+1}$ such that $s_1 \in_v Q$, and for each $j \in \{1, 2, \ldots, n\}$, $s_j \xrightarrow{act_1, act_2, \ldots, act_k}_d s_{j+1}$.

Since the sequence of states $s_2, s_3 \ldots s_{n+1}$ has length n, there exist $i, m \in \{2, 3, \ldots, n+1\}$ such that $i < m$ and $s_i \equiv_{A|v} s_m$. □

Lemma A.10. *Let $Q = (l, [\![C, D]\!])$ be symbolic states in $[\![A]\!]^\alpha$ such that $v \in [\![C]\!]$. If $Q \xRightarrow{act_1, act_2, \ldots, act_k}_\alpha Q$, then there exist concrete states s', s'', s''' in $[\![A]\!]_v$ such that $s' \in_v Q$, $s' \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s''$, $s'' \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s'''$ and $s'' \equiv_{A|v} s'''$.*

Proof. Since $Q \xRightarrow{act_1, act_2, \ldots, act_k}_\alpha Q$, by definition, there exist $s \in_v Q$. By Lemma A.9, there exist $s_1, s_2, s_3, \ldots, s_m$ and $i \in \{2, 3, \ldots, m-1\}$ such that $s_1 \in_v Q$, $s_i \equiv_{A|v} s_m$, and for each $j \in \{1, 2, \ldots, m-1\}$, $s_j \xrightarrow{act_1, act_2, \ldots, act_k}_d s_{j+1}$.

Let $s' = s_1$, $s'' = s_i$, and $s''' = s_m$, then $s' \in_v Q$, $s' \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s''$, $s'' \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s'''$, and $s'' \equiv_{A|v} s'''$. □

Lemma A.11. *Let s_1, s_2 be concrete states in $[\![A]\!]_v$. If $s_1 \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s_2$ and $s_1 \equiv_{A|v} s_2$, then there is an infinite sequence of concrete states $s_1 s_2 \ldots$ in $[\![A]\!]_v$ such that for each $i \geq 1$, $s_i \xrightarrow{act_1, act_2, \ldots, act_k}_d s_{i+1}$.*

Proof. Follows from Lemma A.4. □

Lemma A.12. *Let $Q = (l, [\![C, D]\!])$ be a symbolic state in $[\![A]\!]^\alpha$ such that $v \in [\![C]\!]$. If $Q \xRightarrow{act_1, act_2, \ldots, act_k}_\alpha Q$, then there is an infinite sequence of concrete states $s_1 s_2 \ldots$ in $[\![A]\!]_v$ such that $s_1 \in_v Q$, and for each $i \geq 1$, $s_i \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s_{i+1}$.*

Proof. By Lemma A.10, there exist s_1, s_2, s_3 such that $s_1 \in_v Q$, $s_1 \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s_2$, $s_2 \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s_3$ and $s_2 \equiv_{A|v} s_3$.

By Lemma A.11, there is an infinite sequence of states $s_2, s_3, s_4, \ldots$ such that for each $i \geq 2$, $s_i \xrightarrow{act_1, act_2, \ldots, act_k}_d{}^* s_{i+1}$. □

Theorem A.13. *Let $A = ((L, l_0, X, P, \Delta, Inv), F)$ be a PTBA and α be a finite abstraction. For each parameter valuation v the following holds: there exists an accepting run of $[\![A]\!]_v$ if and only if there exists an accepting v-run of $[\![A]\!]^\alpha$.*

Proof. The fact that the existence of an accepting run of $[\![A]\!]_v$ implies the existence of an accepting v-run of $[\![A]\!]^\alpha$ can be proved easily for each valuation v by induction and Lemma A.6.

We now give the proof for the other direction. If $[\![A]\!]^{\alpha} = (\mathbb{Q}_A, \mathbb{Q}_{init}, \Longrightarrow^{\alpha})$ over L has an accepting v-run, then there exists a $Q = (l, [\![C, D]\!]) \in \mathbb{Q}_A$ and $act_0, act_1, \ldots, act_i, \ldots, act_k \in L$ such that $Q_0 \xrightarrow{act_0, act_1, \ldots, act_{i-1}}_{\alpha} Q$, and $Q \xrightarrow{act_i, act_{i+1}, \ldots, act_k}_{\alpha} Q$, and $F \cap \{act_i, act_{i+1}, \ldots, act_k\} \neq \emptyset$ where Q_0 is the initial state of $[\![A]\!]^{\alpha}$ and $v \in [\![C]\!]$.

Applying Lemma A.12 to $Q_i \xrightarrow{act_i, act_{i+1}, \ldots, act_k}_{\alpha} Q_i$ we have an infinite sequence of states $s'_2 s'_3 s'_4 \ldots$ such that $s'_2 \in_v Q_i$ and for each $j \geq 2$ $s'_j \xrightarrow{act_i, act_{i+1} \ldots, act_k}_{d}{}^{*} s'_{j+1}$.

Applying Lemma A.7 to $Q_0 \xrightarrow{act_0, act_1, \ldots, act_i - 1}_{\alpha} Q_i$ and $s'_2 \in_v Q$, it follows that there exist s', s'' such that $s' \in_v Q_0$, $s' \xrightarrow{act_0, act_1, \ldots, act_{i-1}}_{d} s''$, and $s'_2 \preccurlyeq s''$.

By $s' \in_v Q_0$ and $Q_0 \in \alpha(S_0)$, we know that there exists a $s_1 \in_v S_0$ such that $s' \preccurlyeq s_1$. From the fact that $s' \xrightarrow{act_0, act_1, \ldots, act_{i-1}}_{d} s''$, and $s' \preccurlyeq s_1$, we know that there exists a s_2 such that $s_1 \xrightarrow{act_0, act_1, \ldots, act_{i-1}}_{d} s_2$, and $s'' \preccurlyeq s_2$. Thus we have obtained that $s'_2 \preccurlyeq s_2$.

Applying Lemma A.3 to $s'_2 \preccurlyeq s_2$ and $s'_j \xrightarrow{act_i, act_{i+1} \ldots, act_k}_{d}{}^{*} s'_{j+1}$ $(j = 2, 3, \ldots)$, we can obtain an infinite sequence of states $s_3 s_4 \ldots$ such that $s_2 \xrightarrow{act_i, act_{i+1} \ldots, act_k}_{d}{}^{*} s_3 \xrightarrow{act_i, act_{i+1} \ldots, act_k}_{d}{}^{*} s_4 \ldots$.

Furthermore, from the fact that $s_1 \in_v S_0$ it follows that there is a $d \in \mathbb{R}^{\geq 0}$ such that $s_0 \xrightarrow{d} s_1$ where s_0 is the initial state of $[\![A]\!]_v$.

Thus, we have proved that there exists an infinite sequence of states $s_1, s_2, \ldots$ such that $s_0 \xrightarrow{\delta} s_1 \xrightarrow{act_0, act_1, \ldots, act_{i-1}}_{d} s_2 \xrightarrow{act_i, act_{i+1}, \ldots, act_k}_{d}{}^{*} s_3 \xrightarrow{act_i, act_{i+1}, \ldots, act_k}_{d}{}^{*} s_4 \ldots$. Now, by the fact that $F \cap \{act_i, act_{i+1}, \ldots, act_k\} \neq \emptyset$, we know that $[\![A]\!]_v$ has an infinite accepting run. □

Finally, we provide the proof of Theorem 3.15.

Theorem *(Theorem 3.15). Let A be a PTBA. The pk-extrapolation is a finite abstraction such that $[\![A]\!]^{pk\text{-}extrapolation}$ preserves all accepting runs of $[\![A]\!]_v$ for each parameter valuation v.*

Proof. Follows directly from Lemma A.1 and Theorem A.13. □

Appendix B. Proofs of Lemmata 5.7 and 5.8

Let us assume a fixed PTA $A = (L, l_0, X, P, \Delta, Inv)$, a fixed lb, ub functions, a fixed set of simple non-diagonal guards without parameters $G \subseteq \overline{G}_1(X, \emptyset)$, and a fixed location labelling function $\mathcal{L}$. We start by proving several auxiliary lemmata. In the following we use $U(\eta)$ to denote the ultraregion containing η.

Lemma B.1. *All reachable states $(l, [\![C, D]\!], U)$ of A_{PZURA} satisfy the following invariants: for each $v \in [\![C]\!]$ holds $[\![D]\!]_v \cap U \neq \emptyset$ and $[\![D]\!]_v = [\![D^{\uparrow}]\!]_v \cap [\![Inv(l)]\!]_v$.*

Proof. The lemma follows from the definition of A_{PZURA} and *pk-extrapolation*. □

Lemma B.2. *[9] Let (C, D) be a CPDBM in a canonical form. Then $[\![D]\!]_v \neq \emptyset$ for each $v \in [\![C]\!]$.*

Lemma B.3. *[9] Let (C, D) and (C', D') be a CPDBM and (C', D') is canonical. Then $[\![C', D']\!] \subseteq [\![C, D]\!] \Leftrightarrow ([\![C']\!] \subseteq [\![C]\!] \wedge \forall i, j : C' \models e'_{i,j}(\prec'_{ij} \Rightarrow \prec_{ij}) e_{ij})$.*

Lemma B.4. *[9] $[\![C, D]\!] \cap [\![g]\!] = \bigcup \{[\![C', D']\!] \mid (C', D') \in (C, D)[g]\}$.*

Lemma B.5. *[9] $[\![C, D]\!] = \bigcup \{[\![C', D']\!] \mid (C', D') \in (C, D)_c\}$.*

Lemma B.6. *Let v be a parameter valuation, l a location, U an ultraregion, η a clock valuation, and (C, D) a CPDBM. For each $(v, \eta) \in [\![C, D]\!]$ there exists $(l, [\![C', D']\!], U) \in \alpha_{pk}((l, [\![C, D]\!], U))$ such that $(v, \eta) \in [\![C', D']\!]$.*

Proof. Each $(l, [\![C', D']\!], U) \in \alpha_{pk}((l, [\![C, D]\!], U))$ is computed as a result of $i \cdot j$ successive modifications of the original (C, D). For each modification the following two properties hold. First, consider three versions of C'_{ij} from Definition 5.1 one for each condition. Observe that $[\![e_{ij} > M(x_i)]\!] \cup ([\![-M(x_j) \leq e_{ij}]\!] \cap [\![e_{ij} \leq M(x_i)]\!]) \cup [\![e_{ij} < -M(x_j)]\!] = [\![tt]\!]$ which means no parameter valuation is lost. Second, for each parameter valuation v it holds that $[\![D_{ij}]\!]_v \subseteq [\![D'_{ij}]\!]_v$ which means no clock valuation is lost. Therefore, there exists $(l, [\![C', D']\!], U) \in \alpha_{pk}((l, [\![C, D]\!], U))$ such that $v \in [\![C']\!]$ and $[\![D]\!]_v \subseteq [\![D']\!]_v$. Thus, we also have $(v, \eta) \in [\![C', D']\!]$. □

Lemma B.7. *Let ν be a parameter valuation. Let $(l,\eta) \xrightarrow{act} (l',\eta') \xrightarrow{d} (l',\eta'+d)$ in $[\![A|_\nu]\!]$ such that $\eta' \simeq_G \eta'+d$. Let $(l,[\![C,D]\!],U)$ be a state such that $\nu \in [\![C]\!]$ and $\eta \in [\![D]\!]_\nu \cap U$. Then $(l,[\![C,D]\!],U) \xRightarrow{act} (l',[\![C',D']\!],U')$ with $\nu \in [\![C']\!]$ and $\eta'+d \in [\![D']\!]_\nu \cap U'$.*

Proof. Let $l \xrightarrow{g,R}_\Delta l'$ be the parametric timed automaton transition corresponding to $(l,\eta) \xrightarrow{act} (l',\eta')$. This means that $\eta \in [\![g]\!]_\nu$ and $\eta' = \eta\langle R\rangle$. Let U' be the ultraregion such that $\eta' \in U'$. Clearly, $U' \in U\lfloor R\rfloor$. Using Lemma B.4 and Lemma B.5 we get the existence of $(C_1,D_1) \in (C,D)[U]_c[g]_c$ such that $(\nu,\eta) \in [\![C_1,D_1]\!]$. From the facts $\eta' = \eta\langle R\rangle$, $\eta' \in U'$, Lemma B.4, and Lemma B.5 we now have the existence of $(C_2,D_2) \in (C_1,D_1\langle R\rangle)[U']_c$ such that $(\nu,\eta') \in [\![C_2,D_2]\!]$. As $(l',\eta') \xrightarrow{d} (l',\eta'+d)$, we know that $\eta'+d \in [\![Inv(l')]\!]_\nu$. Therefore, using Lemma B.4 and Lemma B.5 we get the existence of $(C_3,D_3) \in (C_2,D_2^{\uparrow})[Inv(l')]_c$ such that $(\nu,\eta'+d) \in [\![C_3,D_3]\!]$. The fact that $\eta'+d \in U'$ follows from $\eta' \simeq_G \eta'+d$. Then using Lemma B.4 and Lemma B.5 we also obtain the existence of $(C_4,D_4) \in (C_3,D_3)[U']_c$ such that $(\nu,\eta'+d) \in [\![C_4,D_4]\!]$. Obviously, $[\![C_4,D_4]\!] \neq \emptyset$ holds. From Lemma B.3 we get $(\nu,\eta'+d) \in [\![C_4,D_3]\!]$. Now, using Lemma B.6, we get the existence of $(l',[\![C',D']\!],U') \in \alpha_{pk}((l',[\![C_4,D_3]\!],U'))$ such that $(\nu,\eta'+d) \in [\![C',D']\!]$. Finally, we have $(l,[\![C,D]\!],U) \xRightarrow{act} (l',[\![C',D']\!],U')$ with $\nu \in [\![C']\!]$ and $\eta'+d \in [\![D']\!]_\nu \cap U'$. □

Lemma B.8. *Let ν be a parameter valuation. Let $(l,\eta) \xrightarrow{d} (l,\eta+d)$ in $[\![A|_\nu]\!]$ such that $U(\eta+d) = succ(U(\eta))$ and let $(l,[\![C,D]\!],U)$ be a state such that $\nu \in [\![C]\!]$ and $\eta \in [\![D]\!]_\nu \cap U$. Let either $succ(U) \neq U$ or $[\![Inv(l)]\!]_\nu = \mathbf{tt}$. Then $(l,[\![C,D]\!],U) \xRightarrow{\delta} (l,[\![C',D]\!],succ(U))$.*

Proof. The fact that $\eta \in [\![D]\!]_\nu$ together with the second invariant of Lemma B.1 implies that $\eta+d \in [\![D]\!]_\nu$. Therefore $[\![D]\!]_\nu \cap succ(U) \neq \emptyset$. Now, from the fact that $\nu \in [\![C]\!]$, $\eta+d \in [\![D]\!]_\nu \cap succ(U)$, Lemma B.4, and Lemma B.5 we obtain the existence of $(C',D') \in (C,D)[succ(U)]_c$ such that $(\nu,\eta+d) \in [\![C',D']\!]$. The fact that $succ(U) = U$ implies $[\![Inv(l)]\!]_\nu = \mathbf{tt}$. Thus, the second invariant of Lemma B.1 implies that $[\![D]\!]_\nu = [\![D^{\uparrow}]\!]_\nu$. In any case, $(l,[\![C,D]\!],U) \xRightarrow{\delta} (l,[\![C',D]\!],succ(U))$. □

Repeated application of the previous lemma gives the following result.

Lemma B.9. *Let ν be a parameter valuation. Let $(l,\eta) \xrightarrow{d} (l,\eta+d)$ in $[\![A|_\nu]\!]$ and let $(l,[\![C,D]\!],U)$ be a state such that $\nu \in [\![C]\!]$ and $\eta \in [\![D]\!]_\nu \cap U$. Then there exists $i \in \mathbb{N}_0$ such that $(l,[\![C,D]\!],U) \xRightarrow{\delta} (l,[\![C_1,D]\!],succ(U)) \xRightarrow{\delta} \cdots \xRightarrow{\delta} (l,[\![C_i,D]\!],succ^i(U))$ with $\nu \in [\![C_i]\!]$ and $\eta+d \in [\![D]\!]_\nu \cap succ^i(U)$.*

Lemma *(Lemma 5.7). Let ν be a parameter valuation and let π be a proper run of $[\![A|_\nu]\!]$ with signature $sig_{Ap}(\pi)$. Then there exists an infinite ν-run of A_{PZURA} $(l_0,[\![C_0,D_0]\!],U_0) \xRightarrow{\gamma_0} (l_1,[\![C_1,D_1]\!],U_1) \xRightarrow{\gamma_1} \cdots$ with $\gamma_i \in \{act,\delta\}$ such that $sig_{Ap}(\pi)(i) = \mathcal{L}(l_i) \cup G(U_i)$, where $sig_{Ap}(\pi)(i)$ denotes the ith letter (from 2^{Ap}) of $sig_{Ap}(\pi)$.*

Proof. We say that $\rho = S_0 \xRightarrow{\gamma_0} S_1 \xRightarrow{\gamma_1} \ldots$ is a ν-path of $A_{PZURA} = (\mathbb{S}_A, \mathbb{S}_{init}, \Longrightarrow)$ if for each $i \in \mathbb{N}_{>0}$ it holds that $S_i \in \mathbb{S}_A$, $S_i = (l_i,[\![C_i,D_i]\!],U_i)$, $\nu \in [\![C_i]\!]$ and for each $i \in \mathbb{N}_0$ it holds that $S_{i-1} \xRightarrow{\gamma_{i-1}} S_i$. We also say that the state $S = (l,[\![C,D]\!],U)$ ν-contains (l',η') if $l = l'$ and $\eta' \in [\![D]\!]_\nu$. Let $\pi = (l_0,\eta_0) \xrightarrow{d_0} (l_0,\eta_0+d_0) \xrightarrow{act} (l_1,\eta_1)\cdots$. Let $sig_{Ap}(\pi) = w_0w_1w_2\cdots$ where w_i are as given in Definition 5.6.

We inductively define states S_i and finite ν-paths ρ_i of A_{PZURA} and then show how to build the desired ν-run. The intended invariant is that for all parameter valuations ν the state $S_i = (l_i,[\![C_i,D_i]\!],U_i)$ ν-contains (l_i,η_i) and ρ_i connects S_{i-1} to S_i with the following exception: If w_j is the last in $sig_{Ap}(\pi)$ then ρ_j is an infinite ν-path from S_{i-1}. Initially, $S_0 = (l_0,[\![C_0,D_0]\!],U_0)$ and ρ_0 is empty. We now build S_{i+1} and ρ_i for $i \geq 0$.

Assume first that w_i is finite. This means that $(l_i,\eta_i) \xrightarrow{d_i} (l_i,\eta_i+d_i) \xrightarrow{act} (l_{i+1},\eta_{i+1})$ in π. Applying Lemma B.9 to $(l_i,\eta_i) \xrightarrow{d_i} (l_i,\eta_i+d_i)$ and S_i gives a ν-path $S_i \xRightarrow{\delta} \cdots \xRightarrow{\delta} (l_i,[\![C',D']\!],U')$ such that $\eta_i+d_i \in [\![D']\!]_\nu \cap U'$. Applying Lemma B.7 to $(l_i,\eta_i+d_i) \xrightarrow{act} (l_{i+1},\eta_{i+1})$ and $(l_i,[\![C',D']\!],U')$ gives $(l_i,[\![C',D']\!],U') \xRightarrow{act} (l_{i+1},[\![C'',D'']\!],U'')$ such that $\eta_{i+1} \in [\![D'']\!]_\nu \cap U''$. Let $S_{i+1} = (l_{i+1},[\![C'',D'']\!],U'')$ and let ρ_i be the ν-path from S_i to S_{i+1} created by the composition of the two ν-paths above.

Assume now that w_i is infinite. This means that $(l_i,\eta_i) \xrightarrow{\infty} (l,\infty)$ and thus $Inv(l_i) = \mathbf{tt}$. Let $d \in \mathbb{R}_{\geq 0}$ be such that $\eta_i+d \in U$ with $U = succ(U)$. Such d has to exist due to the unbounded invariant of l. Applying Lemma B.9 to $(l_i,\eta_i) \xrightarrow{d} (l_i,\eta_i+d)$ gives a ν-path $S_i \xRightarrow{\delta} \cdots \xRightarrow{\delta} (l_i,[\![C',D']\!],U')$ where $U' = succ(U')$. Applying Lemma B.8 to $(l_i,\eta_i+d) \xrightarrow{0} (l_i,\eta_i+d)$ gives a loop $(l_i,[\![C',D']\!],U') \xRightarrow{\delta} (l_i,[\![C',D']\!],U')$. Combining the ν-path with the loop gives an infinite ν-path, let us denote it by ρ_i.

The resulting ν-run of A_{PZURA} is $S_0\rho_iS_1\rho_1\cdots$, which ends with ρ_i iff $sig_{Ap}(\pi)$ ends with w_i. It is clear that this run satisfies the statement of the lemma. □

We have thus shown that for every v holds that every proper run of $[\![A|_v]\!]$ has its v-run counterpart in A_{PZURA}. However, to show the opposite we lack properties similar to Lemmata B.7, B.8, and B.9. One of the reasons is the extrapolation. However, even in the absence of extrapolation, we could only prove a contravariant version of these lemmata, namely that for each v and for each transition $(l, [\![C, D]\!], U) \overset{\gamma}{\Longrightarrow} (l', [\![C', D']\!], U')$ with $v \in [\![C]\!]$, $v \in [\![C']\!]$ and each $\eta \in [\![D']\!]_v \cap U'$ there exists $\xi \in [\![D]\!]_v \cap U$ and corresponding transitions that lead from (l, ξ) to (l', η). This would allow us to prove that every finite v-path of A_{PZURA} has a counterpart in $[\![A|_v]\!]$, but that is insufficient, as there is an uncountably infinite number of finite paths in $[\![A|_v]\!]$ and we cannot combine them to make one infinite run. We thus use a roundabout way similar to that used in [37] (the differences lie in the existence of delay transitions in A_{PZURA} and in the parametric definition of symbolic states). We use the standard region equivalence to partition the states of $[\![A|_v]\!]$ into regions and show that every v-run of A_{PZURA} corresponds to a run in the region graph of $A|_v$. Again, our region graph is a bit nonstandard to account for the $\overset{\delta}{\Rightarrow}$ transitions.

In the following definition, we use $\lfloor \cdot \rfloor$ to denote the floor function and $\{\cdot\}$ to denote the fractional part function.

Definition B.10 *(Region equivalence).* Let $Max \in \mathbb{N}_0$. Two clock valuations η and η' are region equivalent w.r.t. Max, denoted by $\eta \equiv_{Max} \eta'$, if for all $x, y \in X$

1. $\eta(x) > Max$ iff $\eta'(x) > Max$,
2. if $\eta(x) \leq Max$ then $\lfloor \eta(x) \rfloor = \lfloor \eta'(x) \rfloor$,
3. if $\eta(x) \leq Max$ then $\{\eta(x)\} = \{\eta'(x)\}$, and
4. if $\eta(x) \leq Max$ and $\eta(y) \leq Max$ then $\{\eta(x)\} \leq \{\eta(y)\}$ iff $\{\eta'(x)\} \leq \{\eta'(y)\}$.

The equivalence classes of $\equiv_{Max}$ are called regions.

In the following, let Max be a fixed integer that is greater or equal to all bounds appearing in timed automaton $A|_v$ and all bounds appearing in G. Clearly, every ultraregion is a union of regions w.r.t. Max. Having such Max, all the clock valuations in a given region are indistinguishable by any invariant or any guard appearing in $A|_v$ or G. We may thus extend notation from clock valuations to regions, such as $r \models g$, $U(r)$, and $G(r)$.

A result from [37] says that $\mathcal{E}(Z) \subseteq \{r \mid r \cap Z \neq \emptyset\}$ for a clock zone Z and k-extrapolation $\mathcal{E}$. This implies $r \cap \mathcal{E}(Z) \neq \emptyset \implies r \cap Z \neq \emptyset$. We shall use this fact in the following.

A region automaton of $A|_v$ respecting G is a finite transition system with states of the form (l, r) where l is a location and r a region with $r \models Inv(l)$. We have two kinds of transitions: $(l, r) \xrightarrow{\delta} (l, r')$ if there exists $\eta \in r$ and $d \in \mathbb{R}_{>0}$ such that $(l, \eta) \xrightarrow{d} (l, \eta + d)$, $\eta + d \in r'$, $U(r') = succ(U(r))$ and if $U(r) = U(r')$ then $Inv(l) = \mathbf{tt}$; and $(l, r) \xrightarrow{act} (l', r')$ if there exists $d \in \mathbb{R}_{\geq 0}$, $\eta \in r$, and $\eta' \in r'$ such that $(l, \eta) \xrightarrow{act} (l', \eta' - d) \xrightarrow{d} (l', \eta')$ and $U(\eta' - d) = U(\eta')$.

The following lemma follows from the fact that $\equiv_{Max}$ is a time-abstracting bisimulation [38].

Lemma B.11. *If $(l, r) \xrightarrow{\delta} (l, r')$ then for all $\eta \in r$ there exists $d \in \mathbb{R}_{>0}$ such that $\eta + d \in r'$ and $(l, \eta) \xrightarrow{d} (l, \eta + d)$. If $(l, r) \xrightarrow{act} (l', r')$ then for all $\eta \in r$ there exists $\eta' \in r'$ and $d \in \mathbb{R}_{\geq 0}$ such that $(l, \eta) \xrightarrow{act} (l', \eta' - d) \xrightarrow{d} (l', \eta')$.*

Lemma B.12. *Let v be a parameter valuation, $\gamma \in \{act, \delta\}$, and $(l, [\![C, D]\!], U)$ be a v-reachable state of A_{PZURA} with $(l, [\![C, D]\!], U) \overset{\gamma}{\Longrightarrow} (l', [\![C', D']\!], U')$ and $v \in [\![C']\!]$. Then for each region r' such that $r' \cap [\![D']\!]_v \cap U' \neq \emptyset$ there exists a region r such that $r \cap [\![D]\!]_v \cap U \neq \emptyset$ and $(l, r) \xrightarrow{\gamma} (l', r')$.*

Proof. Let first $\gamma = act$. This means that $l \xrightarrow{g,R}_{\Delta} l'$, $U' \in U\lfloor R \rfloor$ and there exist satisfiable CPDBMs (C_1, D_1), (C_2, D_2), (C_3, D_3), (C_4, D_4) such that $[\![C_4, D_4]\!] \neq \emptyset$, $(l', [\![C', D']\!], U') \in \alpha_{pk}(l', [\![C_4, D_3]\!], U')$, $(C_4, D_4) \in (C_3, D_3)[U']_c$, $(C_3, D_3) \in (C_2, D_2^{\uparrow})[Inv(l')]_c$, $(C_2, D_2) \in (C_1, D_1\langle R \rangle)[U']_c$, and $(C_1, D_1) \in (C, D)[U]_c[g]_c$. Let r' be a region with $r' \cap [\![D']\!]_v \cap U' \neq \emptyset$. The pk-extrapolation can be effectively seen as k-extrapolation over a clock zone when considering a particular parameter valuation. Thus, $[\![D']\!]_v = \mathcal{E}([\![D_3]\!]_v)$.

As explained above, a result from [37] implies that $r' \cap [\![D_3]\!]_v \cap U' \neq \emptyset$. Let us take a clock valuation $\eta \in r' \cap [\![D_3]\!]_v \cap U'$. As $\eta \in [\![D_3]\!]_v$, there has to exist $d \in \mathbb{R}_{\geq 0}$ such that $\eta - d \in [\![D_2]\!]_v$ and $(\eta - d)(x) = 0$ for all $x \in R$. Then there exists $\xi \in [\![D_1]\!]_v$ such that $\xi(x) = (\eta - d)(x)$ for all $x \notin R$. Using Lemma B.4 and Lemma B.5 we have also $\xi \in [\![D]\!]_v$ and $\xi \in U$. Let r be the region that includes ξ. Then $r \cap [\![D]\!]_v \cap U \neq \emptyset$ and $(l, \xi) \xrightarrow{act} (l', \eta - d) \xrightarrow{d} (l', \eta)$. Thus $(l, r) \xrightarrow{act} (l', r')$.

Let now $\gamma = \delta$. This means that $l = l'$, $D = D'$ and $U' = succ(U)$. Clearly, either $(l, [\![C, D]\!], U)$ is reachable from the initial state only via $\overset{\delta}{\Rightarrow}$ transitions or there exists some $(l, [\![\overline{C}, D]\!], \overline{U})$ such that $(l, [\![\overline{C}, D]\!], \overline{U})$ is a result of an $\overset{act}{\Longrightarrow}$ transition and $(l, [\![\overline{C}, D]\!], \overline{U}) \overset{\delta}{\Rightarrow} \cdots \overset{\delta}{\Rightarrow} (l, [\![C, D]\!], U)$.

In the first case, the statement is obvious.

Let now $(l, [\![\overline{C}, D]\!], \overline{U})$ be as described above. This means that there is some l_p, C_p, D_p, U_p such that $l_p \xrightarrow{g,R}_{\Delta} l$, $\overline{U} \in U_p\lfloor R \rfloor$ and there exist satisfiable CPDBMs $(C_1, D_1), (C_2, D_2), (C_3, D_3), (C_4, D_4)$ such that $[\![C_4, D_4]\!] \neq \emptyset$, $(l, [\![\overline{C}, D]\!], \overline{U}) \in \alpha_{pk}(l, [\![C_4, D_3]\!], \overline{U})$, $(C_4, D_4) \in (C_3, D_3)[\overline{U}]_c$, $(C_3, D_3) \in (C_2, D_2^{\uparrow})[Inv(l)]_c$, $(C_2, D_2) \in (C_1, D_1\langle R\rangle)[\overline{U}]_c$, and $(C_1, D_1) \in (C_p, D_p)[U_p]_c[g]_c$. Let now r' be a region such that $r' \cap [\![D]\!]_v \cap U' \neq \emptyset$. Then, as mentioned above, also $r' \cap [\![D_3]\!]_v \cap U' \neq \emptyset$. Let $\eta \in r' \cap [\![D_3]\!]_v \cap U'$. Due to the fact that $(C_3, D_3) \in (C_2, D_2^{\uparrow})[Inv(l)]_c$ (using Lemma B.4, Lemma B.5 and time successors definition), there is some $h \in \mathbb{R}_{\geq 0}$ such that $\eta - h \in [\![D_2]\!]_v$ and also $\eta \in [\![D_3]\!]_v \cap [\![Inv(l)]\!]_v$. We have $\eta - h \in [\![Inv(l)]\!]_v$ as invariant consists of upper bounds only. Thus, also $\eta - h \in [\![D_3]\!]_v$. Now, from the fact that $(C_2, D_2) \in (C_1, D_1\langle R\rangle)[\overline{U}]_c$ (again using Lemma B.4 and Lemma B.5), we obtain $\eta - h \in \overline{U}$. Thus, $\eta - h \in [\![D_3]\!]_v \cap \overline{U}$. We know that $U = succ^i(\overline{U})$ for some i. This means that there exists some d such that $\eta - h + d \in U$. Clearly, $d \leq h$. As zones are convex, also $\eta - h + d \in [\![D_3]\!]_v \subseteq [\![D]\!]_v$. Take the region r such that $\eta - h + d \in r$. Then $r \cap [\![D]\!]_v \cap U \neq \emptyset$ and $(l, \eta - h + d) \xrightarrow{h-d} (l, \eta)$. Thus $(l, r) \xrightarrow{\delta} (l, r')$. □

Lemma B.13. *Let v be a parameter valuation and let $(l_0, [\![C_0, D_0]\!], U_0) \xRightarrow{\gamma_0} (l_1, [\![C_1, D_1]\!], U_1) \xRightarrow{\gamma_1} \cdots$ be a v-run of A_{PZURA} with $\gamma_i \in \{act, \delta\}$. Then there exists a run $(l_0, r_0) \xrightarrow{\gamma_0} (l_1, r_1) \xrightarrow{\gamma_1} \cdots$ of the region automaton of $[\![A|_v]\!]$ such that $r_i \subseteq [\![D_i]\!]_v \cap U_i$.*

Proof. The proof of this lemma is similar to the proof of Theorem 7 in [37]. We construct a directed acyclic graph with nodes (i, q_i, r_i) such that $r_i \subseteq [\![D_i]\!]_v \cap U_i$. There is an edge from (i, q_i, r_i) to $(i+1, q_{i+1}, r_{i+1})$ if $(q_i, r_i) \xrightarrow{\gamma_i} (q_{i+1}, r_{i+1})$. Lemma B.12 ensures that every node in this graph (except for $(0, l_0, r_0)$) has at least one predecessor. As the graph is infinite with finite branching, it has an infinite path. □

Lemma *(Lemma 5.8). Let v be a parameter valuation. Let $(l_0, [\![C_0, D_0]\!], U_0) \xRightarrow{\gamma_0} (l_1, [\![C_1, D_1]\!], U_1) \xRightarrow{\gamma_1} \cdots$ be a v-run of A_{PZURA} with $\gamma_i \in \{act, \delta\}$. Then there exists a proper run π of $[\![A|_v]\!]$ such that $sig_{Ap}(\pi)(i) = \mathcal{L}(l_i) \cup G(U_i)$.*

Proof. Take the region automaton run of Lemma B.13 and create a run of $[\![A|_v]\!]$ by repeated application of Lemma B.11. The resulting run starts with an *act* transition $(l_0, \eta) \xrightarrow{act} (l_1, \xi)$ with $\eta \in [\![D_0]\!]_v \cap U_0$. This means that $\eta(x) = \eta(y)$ for all x, y. We extend the run into a proper run with $(l_0, \mathbf{0}) \xrightarrow{\eta(x)} (l_0, \eta)$. □

We have shown that for each parameter valuation v the proper runs of $[\![A|_v]\!]$ and the v-runs of A_{PZURA} are equivalent in the sense that a signature of a proper run of $[\![A|_v]\!]$ is the sequence of labellings on a v-run of A_{PZURA} and vice versa.

References

[1] E. Clarke, O. Grumberg, D. Peled, Model Checking, The MIT Press, 1999.
[2] R. Alur, D.L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (2) (1994) 183–235.
[3] R. Alur, C. Courcoubetis, D.L. Dill, Model-checking for real-time systems, in: Proceedings of the Fifth Annual Symposium on Logic in Computer Science, LICS '90, IEEE Computer Society, 1990, pp. 414–425.
[4] C. Daws, S. Tripakis, Model checking of real-time reachability properties using abstractions, in: Tools and Algorithms for Construction and Analysis of Systems, in: LNCS, vol. 1384, Springer, 1998, pp. 313–329.
[5] G. Behrmann, A. David, K.G. Larsen, J. Håkansson, P. Pettersson, W. Yi, M. Hendriks, UPPAAL 4.0, in: Third International Conference on the Quantitative Evaluation of Systems, QEST 2006, IEEE, 2006, pp. 125–126.
[6] R. Alur, T.A. Henzinger, M.Y. Vardi, Parametric real-time reasoning, in: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, ACM, 1993, pp. 592–601.
[7] J.S. Miller, Decidability and complexity results for timed automata and semi-linear hybrid automata, in: Hybrid Systems: Computation and Control, in: LNCS, vol. 1790, Springer, 2000, pp. 296–309.
[8] N. Beneš, P. Bezděk, K.G. Larsen, J. Srba, Language emptiness of continuous-time parametric timed automata, in: International Colloquium on Automata, Languages, and Programming, in: LNCS, vol. 9135, Springer, 2015, pp. 69–81.
[9] T. Hune, J. Romijn, M. Stoelinga, F.W. Vaandrager, Linear parametric model checking of timed automata, J. Log. Algebraic Program. 52–53 (2002) 183–220.
[10] L. Bozzelli, S. La Torre, Decision problems for lower/upper bound parametric timed automata, Form. Methods Syst. Des. 35 (2) (2009) 121–151.
[11] A. Jovanović, D. Lime, O.H. Roux, Integer parameter synthesis for real-time systems, IEEE Trans. Softw. Eng. 41 (5) (2015) 445–461.
[12] P. Bezděk, N. Beneš, V. Havel, J. Barnat, I. Černá, On clock-aware LTL properties of timed automata, in: Theoretical Aspects of Computing, ICTAC 2014, in: LNCS, vol. 8687, Springer, 2014, pp. 43–60.
[13] É. André, T. Chatain, L. Fribourg, E. Encrenaz, An inverse method for parametric timed automata, Int. J. Found. Comput. Sci. 20 (5) (2009) 819–836.
[14] R. Alur, T.A. Henzinger, A really temporal logic, J. ACM 41 (1) (1994) 181–204.
[15] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Syst. 2 (4) (1990) 255–299.
[16] R. Alur, T. Feder, T.A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1) (1996) 116–146.
[17] J.S. Ostroff, Temporal Logic for Real-Time Systems, Research Studies Press Advanced Software Development Series, vol. 40, 1989.
[18] E. Harel, O. Lichtenstein, A. Pnueli, Explicit clock temporal logic, in: Proceedings of the Fifth Annual Symposium on Logic in Computer Science, LICS '90, 1990, pp. 402–413.
[19] S. Demri, D. D'Souza, An automata-theoretic approach to constraint LTL, Inf. Comput. 205 (3) (2007) 380–415.
[20] G. Li, Z. Tang, Modelling real-time systems with continuous-time temporal logic, in: Formal Methods and Software Engineering, in: LNCS, vol. 2495, Springer, 2002, pp. 231–236.
[21] R. Alur, P. Madhusudan, Decision problems for timed automata: a survey, in: Formal Methods for the Design of Real-Time Systems, in: LNCS, vol. 3185, Springer, 2004, pp. 1–24.

[22] C. Baier, J.-P. Katoen, Principles of Model Checking, The MIT Press, 2008.
[23] P. Bezděk, N. Beneš, J. Barnat, I. Černá, LTL parameter synthesis of parametric timed automata, in: Software Engineering and Formal Methods, in: LNCS, vol. 9763, Springer, 2016, pp. 172–187.
[24] S. Tripakis, Checking timed Büchi automata emptiness on simulation graphs, ACM Trans. Comput. Log. 10 (3) (2009) 15.
[25] D.L. Dill, Timing assumptions and verification of finite-state concurrent systems, in: Automatic Verification Methods for Finite State Systems, in: LNCS, vol. 407, Springer, 1990, pp. 197–212.
[26] J. Bengtsson, W. Yi, Timed automata: semantics, algorithms and tools, in: Lectures on Concurrency and Petri Nets, in: LNCS, vol. 3098, Springer, 2004, pp. 87–124.
[27] P. Pettersson, Modelling and Verification of Real-Time Systems Using Timed Automata: Theory and Practice, Department of Computer Systems, Uppsala University, 1999.
[28] P. Bouyer, Forward analysis of updatable timed automata, Form. Methods Syst. Des. 24 (3) (2004) 281–320.
[29] L. Lamport, What good is temporal logic?, in: IFIP Congress, vol. 83, 1983, pp. 657–668.
[30] S. Tripakis, S. Yovine, A. Bouajjani, Checking timed Büchi automata emptiness efficiently, Form. Methods Syst. Des. 26 (3) (2005) 267–292.
[31] G. Behrmann, P. Bouyer, K.G. Larsen, R. Pelánek, Lower and upper bounds in zone-based abstractions of timed automata, Int. J. Softw. Tools Technol. Transf. 8 (3) (2006) 204–215.
[32] C. Courcoubetis, M.Y. Vardi, P. Wolper, M. Yannakakis, Memory-efficient algorithms for the verification of temporal properties, Form. Methods Syst. Des. 1 (2/3) (1992) 275–288.
[33] P. Bezděk, N. Beneš, I. Černá, Bounded Parameter Synthesis Proof-Of-Concept Tool for Parametric Timed Automata, https://paradise.fi.muni.cz/parameterSynthesis/index.html, 2018 (Accessed 1 March 2018).
[34] R. Bagnara, P.M. Hill, E. Zaffanella, The parma polyhedra library: toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems, Sci. Comput. Program. 72 (1–2) (2008) 3–21.
[35] P. Gastin, D. Oddoux, Fast LTL to Büchi automata translation, in: Computer Aided Verification, 13th International Conference, CAV 2001, in: LNCS, vol. 2102, Springer, 2001, pp. 53–65.
[36] G. Li, Checking timed Büchi automata emptiness using LU-abstractions, in: J. Ouaknine, F.W. Vaandrager (Eds.), Formal Modeling and Analysis of Timed Systems, in: LNCS, vol. 5813, Springer, 2009, pp. 228–242.
[37] F. Herbreteau, B. Srivathsan, I. Walukiewicz, Efficient emptiness check for timed Büchi automata, Form. Methods Syst. Des. 40 (2) (2012) 122–146.
[38] S. Tripakis, S. Yovine, Analysis of timed systems using time-abstracting bisimulations, Form. Methods Syst. Des. 18 (1) (2001) 25–68.