Liveness of Parameterized Timed Networks *

Benjamin Aminof[1], Sasha Rubin[2], Florian Zuleger[1], and Francesco Spegni[3]

[1] TU Vienna, Austria
[2] Università degli Studi di Napoli "Federico II", Italy
[3] Università Politecnica delle Marche, Ancona, Italy

Abstract. We consider the model checking problem of infinite state systems given in the form of parameterized discrete timed networks with multiple clocks. We show that this problem is decidable with respect to specifications given by B- or S-automata. Such specifications are very expressive (they strictly subsume ω-regular specifications), and easily express complex liveness and safety properties. Our results are obtained by modeling the passage of time using symmetric broadcast, and by solving the model checking problem of parameterized systems of untimed processes communicating using k-wise rendezvous and symmetric broadcast. Our decidability proof makes use of automata theory, rational linear programming, and geometric reasoning for solving certain reachability questions in vector addition systems; we believe these proof techniques will be useful in solving related problems.

1 Introduction

Timed automata — finite state automata enriched by a finite number of dense- or discrete-valued clocks — can be used to model more realistic circuits and protocols than untimed systems [3,7]. A timed network consists of an arbitrary but fixed number of timed automata running in parallel [2,1]. In each computation step, either some fixed number of automata synchronize by a rendezvous-transition or time advances. We consider the *parameterized model-checking problem (PMCP)* for timed networks: Does a given specification (usually given by a suitable automaton) hold *for every system size*? Apart from a single result which deals with much weaker synchronization than rendezvous [13], no positive PMCP results for liveness specifications of timed automata are known.

System model: In this paper we prove the decidability of the PMCP for discrete timed networks with no controller and liveness specifications. To do this, we reduce the PMCP of these timed networks to the PMCP of *RB-systems* — systems of finite automata communicating via *k-wise rendezvous* and *symmetric*

broadcast. This broadcast action is symmetric in the sense that there is no designated sender. In contrast, the standard broadcast action can distinguish between sender and receivers, and so the PMCP of liveness properties is undecidable even in the untimed setting [9].

Our Techniques and Results: Classical automata (e.g., nondeterministic Büchi word automata (NBW)) are not able to capture the behaviors of RB-systems. Thus, our decidability result uses nondeterministic BS-automata (and their fragments B- and S-automata) which strictly subsume NBW [6].

We show that the PMCP is decidable for controllerless discrete timed networks and (and systems communicating via k-wise rendezvous and symmetric broadcast) and specifications given by B-automata or S-automata (and in particular by NBW) or for negative specifications (i.e., the set of bad executions) given by BS-automata. We prove decidability by constructing a B-automaton that precisely characterizes the runs of a timed network from the point of view of a single process. Along the way, we also obtain an ExpSpace upper bound for the PMCP of safety properties of discrete timed networks.

In order to build the B-automaton, an intricate analysis of the interaction between the transitions caused by the passage of time (modeled by broadcasts) which involve all processes, and those that are the result of rendezvousing processes, is needed. It is this interaction that makes the problem complicated. Thus, for example, results concerning pairwise rendezvous without broadcast [11] do not extend to our case. Our solution to this problem involves the introduction of the idea of a *rational relaxation* of a Vector Addition System, and geometric lemmas concerning paths in these relaxations. It is important to note that these vector addition systems can not capture the edges that correspond to the passage of time. However, they provide the much needed flexibility in capturing what happens in between time ticks *in the presence of* these ticks.

Related Work. Discrete timed networks with rendezvous and a controller were introduced in [1] where it was shown that safety is decidable using the technique of well-structured transition systems. Their result implies a non-elementary upper bound (which we improve to ExpSpace) for the complexity of the PMCP of safety properties of timed networks without a controller. PMCP of liveness properties for continuous-time networks with a controller process is undecidable [2]. However, their proof heavily relies on time being dense and on the availability of a distinguished controller process. RB-systems with a controller were introduced in [12] where it is proved that under an additional strong restriction on the environment and process templates (called a shared simulation), such systems admit cutoffs that allow one to model check epistemic-temporal logic of the parameterised systems. The main difference between our work and theirs is: we do not have a controller, we make no additional restrictions, and we can model check specifications given by B- or S-automata. The authors in [13] proved that the PMCP is decidable for continuous timed networks synchronizing using conjunctive Boolean guards and MITL and TCTL specifications. Finally, there are many decidability and undecidability results in the untimed setting, e.g.,[14,9,8,4,5].

2 Definitions and Preliminaries

Labeled Transition Systems. A *(edge-)labeled transition system (LTS)* is a tuple $\langle S, I, R, \Sigma \rangle$, where S is the set of *states* (usually $S \subseteq \mathbb{N}$), $I \subseteq S$ are the *initial states*, $R \subseteq S \times \Sigma \times S$ is the *edge relation*, and Σ is the *edge-labels alphabet*. *Paths* are sequences of transitions, and runs are paths starting in initial states.

Automata. We use standard notation and results of automata, such as nondeterministic Büchi word automata (NBW) [15]. A *BS-word automaton (BSW)* ([6]) is a tuple $\langle \Sigma, Q, Q_0, \Gamma, \delta, \Phi \rangle$ where Σ is a finite *input alphabet*, Q is a set of *states*, $Q_0 \subseteq Q$ is a set of *initial states*, Γ is a set of *counter (names)*, $\delta \subseteq Q \times \Sigma \times \mathcal{C}^* \times Q$ is the *transition relation* where $\mathcal{C}$ is the set of *counter operations*, i.e. $c := 0, c := c + 1, c := d$ for $c, d \in \Gamma$, and Φ is the *acceptance condition* described below. A run ρ is defined like for nondeterministic automata over infinite words by ignoring the $\mathcal{C}^*$ component. Denote by $c(\rho, i)$ the ith value assumed by counter $c \in \Gamma$ along ρ. The acceptance condition Φ is a positive Boolean combination of the following conditions ($q \in Q, c \in \Gamma$): (i) q is visited infinitely often (Büchi-condition); (ii) $\limsup_i c(\rho, i) < \infty$ (B-condition); (iii) $\liminf_i c(\rho, i) = \infty$ (S-condition). An automaton that does not use B-conditions is called an *S-automaton (SW)*, and one that does not use S-conditions is called a *B-automaton (BW)*.

It is known that BSWs are relatively well behaved [6]: their emptiness problem is decidable; they are closed under union and intersection, but not complement; and BW (resp. SW) can be complemented to SW (resp. BW). Since BSWs are not closed under complement, we are forced, if we are to use the automata-theoretic approach for model checking (cf. [15]), to give the specification in terms of the undesired behaviours, or to consider specifications in terms of BWs or SWs (which both strictly extend ω-regular languages).

Rendezvous with Symmetric Broadcast (RB-System). Intuitively, RB-systems describe the parallel composition of $n \in \mathbb{N}$ copies of a process *template*. An RB-system evolves nondeterministically: either a k-wise rendezvous action is taken, i.e., k different processes instantaneously synchronize on a rendezvous action a, or the symmetric broadcast action is taken, i.e., all processes must take an edge labeled by $\mathfrak{b}$. Systems without the broadcast action are called R-systems.

In the rest of the paper, fix k (the number of processes participating in a rendezvous), a finite set Σ_{actn} of *rendezvous actions*, the *rendezvous alphabet* $\Sigma_{\mathsf{rdz}} = \cup_{\mathsf{a} \in \Sigma_{\mathsf{actn}}} \{\mathsf{a}_1, \ldots, \mathsf{a}_k\}$, and the *communication alphabet* Σ_{com} which is the union $\{((i_1, \mathsf{a}_1), \ldots, (i_k, \mathsf{a}_k)) \mid \mathsf{a} \in \Sigma_{\mathsf{actn}}, i_j \in \mathbb{N}, j \in [k]\} \cup \{\mathfrak{b}\}$.

A *process template* (or *RB-template*) is a finite LTS $P = \langle S, I, R, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\} \rangle$ such that for every state $s \in S$ there is a transition $(s, \mathfrak{b}, s') \in R$ for some $s' \in S$. We call edges labeled by $\mathfrak{b}$ *broadcast edges*, and the rest *rendezvous edges*. For ease of exposition, we assume (with one notable exception, namely $P^{-\circ}$ defined in Section 3) that for every $\varsigma \in \Sigma_{\mathsf{rdz}}$ there is at most one edge in P labeled by ς

and we denote it by $\text{edge}(\varsigma)$.[4] . The *RB-system* $\mathcal{P}^n$ is defined, given a template P and $n \in \mathbb{N}$, is defined as the finite LTS $\langle Q^n, Q_0^n, \Delta^n, \Sigma_{\mathsf{com}} \rangle$[5] where:

1. Q^n is the set of functions (called *configurations*) of the form $f : [n] \to S$. We call $f(i)$ the *state of process i* in f. Note that we sometimes find it convenient to consider a more flexible naming of processes in which we let Q^n be the set of functions $f : X \to S$, where $X \subset \mathbb{N}$ is some set of size n.
2. The set of *initial configurations* $Q_0^n = \{f \in Q^n \mid f(i) \in I \text{ for all } i \in [n]\}$ consists of all configurations which map all processes to initial states of P.
3. The set of *global transitions* Δ^n are tuples $(f, \sigma, g) \in Q^n \times \Sigma_{\mathsf{com}} \times Q^n$ where one of the following two conditions hold:
   - $\sigma = \flat$, and for every $i \in [n]$ we have that $(f(i), \flat, g(i)) \in R$. This is called a *broadcast transition*.
   - $\sigma = ((i_1, \mathsf{a}_1), \ldots, (i_k, \mathsf{a}_k))$, where $\mathsf{a} \in \Sigma_{\mathsf{actn}}$ is the *action* taken, and $\{i_1, \ldots, i_k\} \subseteq [n]$ are k different processes. In this case, for every $1 \leq j \leq k$ we have that $(f(i_j), \mathsf{a}_j, g(i_j)) \in R$; and $f(i) = g(i)$ for every $i \notin \{i_1, \ldots, i_k\}$. This is called a *rendezvous transition*, and the processes in the set $\text{prcs}(\sigma) := \{i_1, \ldots, i_k\}$ are called the *rendezvousing processes*.

We denote the *action taken* on a global transition $t = (f, \sigma, g)$ by $\mathsf{actn}(t)$. Thus, $\mathsf{actn}(t) := \mathsf{a}$ if $\sigma = ((i_1, \mathsf{a}_1), \ldots, (i_k, \mathsf{a}_k))$, and otherwise $\mathsf{actn}(t) := \flat$.

A process template P induces the *infinite RB-system* $\mathcal{P}$, i.e., the LTS $\mathcal{P} = \langle Q, Q_0, \Delta, \Sigma_{\mathsf{com}} \rangle$ where $Q = \cup_{n \in \mathbb{N}} Q^n$, $Q_0 = \cup_{n \in \mathbb{N}} Q_0^n$, $\Delta = \cup_{n \in \mathbb{N}} \Delta^n$.

Executions of an RB-System, and the Parameterized Model-Checking Problem. Given a global transition $t = (f, \sigma, g)$, and a process i, we say that i *moved* in t iff: $\sigma = \flat$, or $i \in \text{prcs}(\sigma)$. We write $edge_i(t)$ for the edge of P taken by process i in the transition t, and $\bot$ if i did not move in t. Thus, if $\sigma = \flat$ then $edge_i(t) := (f(i), \flat, g(i))$; and if $\sigma = ((i_1, \mathsf{a}_1), \ldots, (i_k, \mathsf{a}_k))$ then $edge_i(t) := (f(i), \mathsf{a}_j, g(i))$ if $\sigma(j) = (i, \mathsf{a}_j)$ for some $j \in [k]$, and otherwise $edge_i(t) := \bot$. Take an RB-System $\mathcal{P}^n = \langle Q^n, Q_0^n, \Delta^n, \Sigma_{\mathsf{com}} \rangle$, a path $\pi = t_1 t_2 \ldots$ in $\mathcal{P}^n$, and a process i in $\mathcal{P}^n$. Define $\text{proj}_\pi(i) := edge_i(t_{j_1}) edge_i(t_{j_2}) \ldots$, where $j_1 < j_2 < \ldots$ are all the indices j for which $edge_i(t_j) \neq \bot$. Intuitively, $\text{proj}_\pi(i)$ is the path in P taken by process i during the path π. Define the set of *executions* $\text{EXEC}_\mathcal{P}$ of $\mathcal{P}$ to be the set of the runs of $\mathcal{P}$ projected onto a single process. Note that, due to symmetry, we can assume w.l.o.g. that the runs are projected onto process 1. Formally, $\text{EXEC}_\mathcal{P} = \{\text{proj}_\pi(1) \mid \pi \text{ is a run of } \mathcal{P}\}$. We denote by $\text{EXEC}_\mathcal{P}^{fin}$ (resp. $\text{EXEC}_\mathcal{P}^\infty$) the finite (infinite) executions in $\text{EXEC}_\mathcal{P}$.

For specifications $\mathcal{F}$ (e.g., LTL, NFWs) interpreted over infinite (resp. finite) words over the alphabet $S \times (\Sigma_{\mathsf{rdz}} \cup \{\flat\}) \times S$ of transitions,[6] the *Parameterized Model Checking Problem* (PMCP) for $\mathcal{F}$ is to decide, given a template P, and a specification $\varphi \in \mathcal{F}$, if all executions in $\text{EXEC}_\mathcal{P}^\infty$ (resp. $\text{EXEC}_\mathcal{P}^{fin}$) satisfy φ.

---

[4] This can always be assumed by increasing the size of the rendezvous alphabet.
[5] Even though Σ_{com} is infinite, Δ^n refers only to a finite subset of it.
[6] In this way we can also capture atomic propositions on edges or states since these atoms may be pushed into the rendezvous label.

Discrete Timed Networks. We refer the reader to [1] for a formal definition of timed networks. Here we describe the templates and informally describe the semantics. Fix a set C of *clocks*. A *timed network template* is a finite LTS $\langle Q, I, R, \Sigma_{\mathsf{rdz}} \rangle$. We associate to each letter $\mathsf{a}_i \in \Sigma_{\mathsf{rdz}}$ a *command* $r(\mathsf{a}_i) \subseteq C$ and a *guard* $p(\mathsf{a}_i)$. A guard p is a Boolean combination of predicates of the form $c \bowtie x$ where $c \in \mathbb{N}$ is a constant, $x \in C$ is a clock, and $\bowtie \in \{<, =\}$.

Intuitively, a discrete timed network consists of the parallel composition of $n \in \mathbb{N}$ template processes, each running a copy of the template. Each copy has a local state (q, t), where $q \in Q$ and $t : C \to \mathbb{N}$. A rendezvous action a is *enabled* if there are k processes in local states (q_i, t_i) ($i \in [k]$) and there are edges $(q_i, \mathsf{a}_i, q_i') \in R$ such that the clocks t_i satisfy the guards $p(\mathsf{a}_i)$. The rendezvous action is *taken* means that the k processes change state (to q_i') and each of the clocks in $r(\mathsf{a}_i)$ is reset to 0. The network evolves non-deterministically, in steps: either all clocks advance by one time unit (so every $t(c)$ increases by one)[7] or a rendezvous action $\mathsf{a} \in \Sigma_{\mathsf{rdz}}$ is taken. For a timed network template T let $\mathcal{T}^n$ denote the timed network composed of $n \in \mathbb{N}$ templates T and let $\mathcal{T}$ denote the union of the networks $\mathcal{T}^n$ for $n \in \mathbb{N}$.

Given a timed network template T one can build an equivalent RB-template P, i.e., $\mathrm{EXEC}_{\mathcal{P}} = \mathrm{EXEC}_{\mathcal{T}}$. The key insight is that the passage of time, that causes all clocks to advance by one time unit, is simulated by symmetric broadcast, and timed-guards are pushed into the template states. The RB-system $\mathcal{P}$ requires only a finite number of states since clock values bigger than the greatest constant appearing on the guards are collapsed to a single abstract value (cf. [1]).

Useful lemmas. We state a few simple but useful lemmas. The first lemma states that, by partitioning processes of an RB-system into independent groups, a system with many processes can simulate in a single run multiple runs of smaller systems. If the simulated paths contain no broadcasts then the transitions of the simulated paths can be interleaved in any order. Otherwise, all simulated runs must have the same number of broadcasts, and the simulations of all the edges before the i'th broadcast on each simulated path must complete before taking the i'th broadcast on the simulating combined path.

Lemma 1 (RB-System Composition). *A system $\mathcal{P}^n$ can, using a single run, partition its processes into groups each simulating a run of a smaller system. All simulated paths must have the same number of broadcasts.*

Consider now an RB-system $\mathcal{P}^n$, and two configurations f, f' in it such that the number of processes in each state in f is equal to that in f', i.e., such that $|f^{-1}(s)| = |f'^{-1}(s)|$ for every $s \in S$. We call f, f' *twins*. A finite path π of length m for which $\mathsf{src}(\pi_1)$ and $\mathsf{dst}(\pi_m)$ are twins is called a *pseudo-cycle*. For example, for P in Figure 1, the following path in $\mathcal{P}^4$ is a pseudo-cycle that is not a cycle:
$$(p,q,q,r) \xrightarrow{((3,\mathsf{c}_1),(4,\mathsf{c}_2))} (p,q,r,p) \xrightarrow{((2,\mathsf{c}_1),(3,\mathsf{c}_2))} (p,r,p,p) \xrightarrow{((3,\mathsf{a}_1),(4,\mathsf{a}_2))} (p,r,q,q).$$

Lemma 2. *By renaming processes after each iteration, a pseudo-cycle π can be pumped to an infinite path which repeatedly goes through the actions on π.*

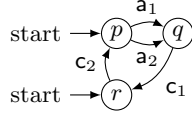[7] Alternatively, as in [1], one can let time advance by any amount.
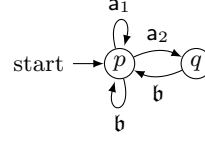
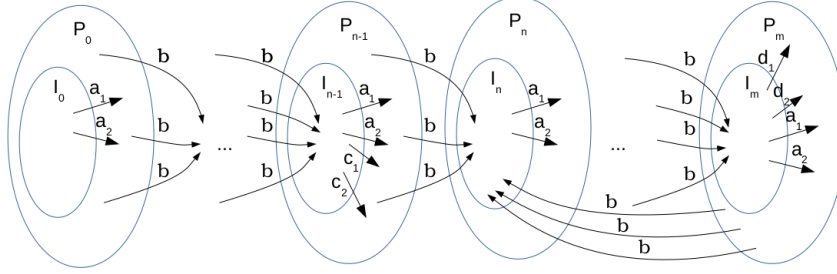Fig. 1: R-template with $k = 2$.   Fig. 2: RB-template with $k = 2$.

Fig. 3: A high level view of the reachability-unwinding lasso.

3   The Reachability-Unwinding of a Process Template

Given template $P = \langle S, I, R, \Sigma_{\mathsf{rdz}} \cup \{\flat\} \rangle$, our goal in this section is to construct a new process template $P^{\rightharpoonup\circ} = \langle S^{\rightharpoonup\circ}, I^{\rightharpoonup\circ}, R^{\rightharpoonup\circ}, \Sigma_{\mathsf{rdz}} \cup \{\flat\} \rangle$, called the *reachability-unwinding* of P, see Figure 3. The template $P^{\rightharpoonup\circ}$ will play a role in all our algorithms for solving the PMCP of RB-systems. Intuitively, $P^{\rightharpoonup\circ}$ is obtained by alternating the following two operations: *(i)* taking a copy of P and removing from it all unreachable rendezvous edges; and *(ii)* unwinding on broadcast edges. This is repeated until a copy is created which is equal to a previous one, we then stop and close the unwinding back into the old copy, forming a high-level lasso structure.

Technically, it is more convenient to first calculate all the desired copies and then to arrange them in the lasso. Thus, we first calculate, for $0 \leq i \leq m$ (for an appropriate m), an R-template $P_i = \langle S_i, I_i, R_i, \Sigma_{\mathsf{rdz}} \rangle$ which is a copy of P with initial states redesignated and all broadcast edges, plus some rendezvous edges, removed. Second, we take $P_0, \ldots, P_m$ and combine them, to create the single process template $P^{\rightharpoonup\circ}$, by connecting the states in P_i with the initial states of P_{i+1} (P_n for $i = m$, where $n \leq m$ is determined by the lasso structure) with broadcast edges, as naturally induced by P.

Construct the R-template $P_i = \langle S_i, I_i, R_i, \Sigma_{\mathsf{rdz}} \rangle$ (called the i'th *component* of $P^{\rightharpoonup\circ}$) recursively: for $i = 0$, we let $I_0 := I$; and for $i > 0$ we let $I_i := \{s \in S \mid (h, \flat, s) \in R$ for some $h \in S_{i-1}\}$ be the set of states reachable from S_{i-1} by a broadcast edge. The elements S_i and R_i are obtained using the following *saturation* algorithm, which is essentially a breadth-first search: start with $S_i := I_i$ and $R_i := \emptyset$; at each round of the algorithm, consider in turn each edge $e = (s, \mathsf{a}_h, t) \in R \backslash R_i$; if for every $l \in [k]$ there is some edge $(s', \mathsf{a}_l, t') \in R$ with $s' \in S_i$, then add e to R_i and add t (if not already there) to S_i. The algorithm ends when

a fixed-point is reached. Observe a property of this algorithm: if $(s, \mathsf{a}_h, t) \in R_i$ then for all $l \in [k] \setminus \{h\}$ there exists $s', t' \in S_i$ such that $(s', \mathsf{a}_l, t') \in R_i$.

Now, P_i is completely determined by I_i (and P), and so there are at most $2^{|S|}$ possible values for it. Hence, for some $n \leq m < 2^{|S|}$ it must be that $P_n = P_{m+1}$. We stop calculating P_i's when this happens since for every $i \in \mathbb{N}_0$ it must be that $P_i = P_{n+((i-n) \mod r)}$, where $r = m + 1 - n$. We call n the *prefix length* of $P^{\multimap}$ (usually denoted by ψ), call r the *period* of $P^{\multimap}$, and for $i \in \mathbb{N}_0$, call $n + ((i - n) \mod r)$ the *associated component number* of i, and denote it by $\text{comp}(i)$.

We now construct from $P_0, \ldots, P_m$ the template $P^{\multimap} = \langle S^{\multimap}, I^{\multimap}, R^{\multimap}, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\} \rangle$, as follows: *(i)* $S^{\multimap} := \cup_{i=0}^m (S_i \times \{i\})$; *(ii)* $I^{\multimap} := I_0 \times \{0\}$ (recall that we also have $I_0 = I$); *(iii)* $R^{\multimap}$ contains the following transitions: the rendezvous transitions $\cup_{i=0}^m \{((s, i), \varsigma, (t, i)) \mid (s, \varsigma, t) \in R_i\}$, and the broadcast transitions $\cup_{i=0}^{m-1} \{((s, i), \mathfrak{b}, (t, i + 1)) \mid (s, \mathfrak{b}, t) \in R \text{ and } s \in S_i\}$ and $\{((s, m), \mathfrak{b}, (t, n)) \mid (s, \mathfrak{b}, t) \in R \text{ and } s \in S_m\}$.

We will abuse notation, and talk about the component P_i, referring sometimes to P_i as defined before (i.e., without the annotation with i), and sometimes to the part of $P^{\multimap}$ that was obtained by annotating the elements of P_i with i.

Observe that, by projecting out the component numbers (we will denote this projecting by superscript $\circledcirc$) from states in $P^{\multimap}$ (i.e., by replacing $(s, i) \in S^{\multimap}$ with $s \in S$), states and transitions in $P^{\multimap}$ induce states and transitions in P. Similarly, paths and runs in $\mathcal{P}^{\multimap}$ can be turned into paths and runs in $\mathcal{P}$. We claim that also the converse is true, i.e., that by adding component numbers, states and transitions in P can be lifted to ones in $P^{\multimap}$; and that by adding the correct (i.e., reflecting the number of previous broadcasts) component numbers to the states of the transitions of a run in $\mathcal{P}$, it too can be lifted to a run in $\mathcal{P}^{\multimap}$. However, a path in $\mathcal{P}$ that is not a run (i.e., that does not start at an initial configuration), may not always be lifted to a path in $\mathcal{P}^{\multimap}$ due to the removal of unreachable edges in the components making up $P^{\multimap}$.

The next lemma says that we may work with template $P^{\multimap}$ instead of P.

Lemma 3. *For every $n \in \mathbb{N}$, we have that $\text{runs}(\mathcal{P}^n) = \{\rho^{\circledcirc} \mid \rho \in \text{runs}((\mathcal{P}^{\multimap})^n)\}$.*

The following lemma says, intuitively, that for every component P_i there is a run of $\mathcal{P}^{\multimap}$ that "loads" arbitrarily many processes into every state of P_i.

Lemma 4. *For all $b, n \in \mathbb{N}$ there is a finite run π of $\mathcal{P}^{\multimap}$ with b broadcasts, s.t., $|f^{-1}(s)| \geq n$ for all states s in the component $P_{comp(b)}$, where $f = \mathsf{dst}(\pi)$.* $\quad\square$

The following lemma states that the set of finite executions of the RB-system $\mathcal{P}$ is equal to the set of finite *runs of the process template $P^{\multimap}$* (modulo component numbers). This is very convenient since, whereas $\mathcal{P}$ is infinite, $P^{\multimap}$ is finite. Unfortunately, when it comes to infinite executions of $\mathcal{P}$ we only get that they are contained in (though in many cases not equal to) the set of infinite runs of $P^{\multimap}$. This last observation is also true for P: consider for example Figure 2 without the $\mathfrak{b}$ edges, and an infinite repetition of the self loop.

Lemma 5. $\text{EXEC}_{\mathcal{P}}^{fin} = \{\pi^{\circledcirc} \mid \pi \in \text{runs}(P^{\multimap}), |\pi| \in \mathbb{N}\}$; *and* $\text{EXEC}_{\mathcal{P}}^{\infty} \subseteq \{\pi^{\circledcirc} \mid \pi \in \text{runs}(P^{\multimap}), |\pi| = \infty\}$

Solving PMCP for regular specifications. Given $P = \langle S, I, R, \Sigma_{\mathsf{rdz}} \cup \{\flat\} \rangle$, let $\mathcal{A}_{\mathcal{P}}^{fin}$ denote the reachability-unwinding $P^{-\circ}$ viewed as an automaton (NFW), with all states being accepting states, and transitions e are labeled $e^{\circledcirc}$ (i.e., they have the component number removed). Formally, $\mathcal{A}_{\mathcal{P}}^{fin} = \langle R, S^{-\circ}, I^{-\circ}, R', S^{-\circ} \rangle$, so the input alphabet of $\mathcal{A}_{\mathcal{P}}^{fin}$ is R (the transition relation of P), and $R' := \{(s, (s^{\circledcirc}, \sigma, t^{\circledcirc}), t) \mid (s, \sigma, t) \in R^{-\circ}\} \subseteq S^{-\circ} \times R \times S^{-\circ}$. Hence:

Theorem 1. *The PMCP of RB-systems (resp. discrete timed networks) for regular specifications is in* PSPACE *(resp.* EXPSPACE*)*

4   Solving PMCP of Liveness Specifications

In this section we show how to solve the PMCP for specifications concerning infinite executions. We begin with the following lemma showing that, if we want to use the automata theoretic approach, classical automata models (e.g. Büchi, Parity) are not up to the task.

Lemma 6. *There is a process template P such that* EXEC$_{\mathcal{P}}^{\infty}$ *is not ω-regular.*

Proof. Consider the process template given in Figure 2. It is not hard to see that in every infinite run of $\mathcal{P}^{n}$ there may be at most $n-1$ consecutive rendezvous transitions before a broadcast transition, resetting all processes to state 1, is taken. Overall, we have that EXEC$_{\mathcal{P}}^{\infty}$ is the set of words of the form $\mathsf{a}_{1}^{n_1} \mathsf{a}_{2}^{m_1} \flat$ $\mathsf{a}_{1}^{n_2} \mathsf{a}_{2}^{m_2} \flat \ldots$, where $m_i \in \{0, 1\}$ for every i, and $\limsup n_i < \infty$. This language is not ω-regular since the intersection of its complement with $\{\mathsf{a}_1, \flat\}^{\omega}$ is not ω-regular (because it contains no ultimately periodic words). $\qquad\square$

In light of Lemma 6, we turn our attention to a stronger model, called BSW [6]. Thus, we solve the PMCP for liveness specifications as follows: given a process template P, we show how to build a BSW $\mathcal{A}_{\mathcal{P}}^{\infty}$ accepting exactly the executions in EXEC$_{\mathcal{P}}^{\infty}$. Model checking of a specification given by a BSW $\mathcal{A}'$ accepting all undesired (i.e., bad) executions, is thus reduced to checking for the emptiness of the intersection of $\mathcal{A}_{\mathcal{P}}^{\infty}$ and $\mathcal{A}'$.

Defining the Automaton $\mathcal{A}_{\mathcal{P}}^{\infty}$. We now describe the structure of the BSW $\mathcal{A}_{\mathcal{P}}^{\infty}$ (in fact we define a BW) accepting exactly the executions in EXEC$_{\mathcal{P}}^{\infty}$.

An important element in the construction is a classification of the edges in $P^{-\circ}$ into four types: blue, green, orange, and red. The red edges are those that appear at most finitely many times on any execution in EXEC$_{\mathcal{P}}^{\infty}$. An edge is blue if it appears infinitely many times on some execution in EXEC$_{\mathcal{P}}^{\infty}$ with finitely many broadcasts, but only finitely many times on every execution which has infinitely many broadcasts. An edge e is green if there is some run $\pi \in$ EXEC$_{\mathcal{P}}^{\infty}$ with infinitely many broadcasts on which e appears unboundedly many times between broadcasts, i.e., if for every $n \in \mathbb{N}$ there are $i < j \in \mathbb{N}$ such that $\pi_i \ldots \pi_j$ contains n occurrences of e and no broadcast edges. An edge which is neither blue, green, nor red is orange. By definition, blue and green edges are not broadcast edges. Since the set EXEC$_{\mathcal{P}}^{\infty}$ is infinite, it is not at all clear that the

problem of determining the type of an edge is decidable. Indeed, this turns out to be a complicated question, and we dedicate Section 4.1 to show that one can decide the type of an edge.

The automaton $\mathcal{A}_{\mathcal{P}}^{\infty}$ is made up of three copies of $\mathcal{A}_{\mathcal{P}}^{fin}$ (called $\mathcal{A}_{\mathcal{P}}^{\infty 1}$, $\mathcal{A}_{\mathcal{P}}^{\infty 2}$, $\mathcal{A}_{\mathcal{P}}^{\infty 3}$), as follows: $\mathcal{A}_{\mathcal{P}}^{\infty 1}$ is an exact copy of $\mathcal{A}_{\mathcal{P}}^{fin}$; the copy $\mathcal{A}_{\mathcal{P}}^{\infty 2}$ has only the green and orange edges left; and $\mathcal{A}_{\mathcal{P}}^{\infty 3}$ has only the blue and green edges left (and in particular has no broadcast edges). Furthermore, for every edge (s, σ, s') in $\mathcal{A}_{\mathcal{P}}^{\infty 1}$ we add two new edges, both with the same source as the original edge, but one going to the copy of s' in the copy $\mathcal{A}_{\mathcal{P}}^{\infty 2}$, and one to the copy of s' in the copy $\mathcal{A}_{\mathcal{P}}^{\infty 3}$. The initial states of $\mathcal{A}_{\mathcal{P}}^{\infty}$ are the initial states of $\mathcal{A}_{\mathcal{P}}^{\infty 1}$. For the acceptance condition: every state in $\mathcal{A}_{\mathcal{P}}^{\infty 2}$ and $\mathcal{A}_{\mathcal{P}}^{\infty 3}$ is a Büchi-state, and there is a single counter $C \in \Gamma_B$ that is incremented whenever an orange rendezvous edge is taken in $\mathcal{A}_{\mathcal{P}}^{\infty 2}$ and reset if a broadcast edge is taken in $\mathcal{A}_{\mathcal{P}}^{\infty 2}$.

Formally, given a process template $P = \langle S, I, R, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\} \rangle$ and its unwinding $P^{\multimap} = \langle S^{\multimap}, I^{\multimap}, R^{\multimap}, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\} \rangle$ define $\mathcal{A}_{\mathcal{P}}^{\infty} = \langle \Sigma, Q, Q_0, \Gamma, \delta, \Phi \rangle$ as:

- The input alphabet Σ is the edge relation R of template P.
- The state set Q is $\{(i, s) \mid s \in S^{\multimap}, i \in \{1, 2, 3\}\}$.
- The initial state set Q_0 is $\{(1, s) \mid s \in I^{\multimap}\}$.
- There is one counter, $\Gamma = \{c\}$.
- The transition relation δ is $\delta_1 \cup \delta_2 \cup \delta_3$, where: δ_1 consists of all tuples $((1, s_1), (s_1^{\circledcirc}, \sigma, s_2^{\circledcirc}), \epsilon, (i, s_2))$ such that $(s_1, \sigma, s_2) \in R^{\multimap}, i \in \{1, 2, 3\}$; and δ_3 consists of all tuples $\{((3, s_1), (s_1^{\circledcirc}, \sigma, s_2^{\circledcirc}), \epsilon, (3, s_2))$ such that $(s_1, \sigma, s_2) \in R^{\multimap}$ is blue or green; and δ_2 consists of all tuples $((2, s_1), (s_1^{\circledcirc}, \sigma, s_2^{\circledcirc}), upd^{\sigma, \rho}, (2, s_2))$ such that $\rho := (s_1, \sigma, s_2) \in R^{\multimap}$ is green or orange, and $upd^{\sigma, \rho}$ is the single operation $c := 0$ if ρ is orange and $\mathsf{actn}(\sigma) = \mathfrak{b}$, and $upd^{\sigma, \rho}$ is the single operation $c := c + 1$ if ρ is orange and $\mathsf{actn}(\sigma) \neq \mathfrak{b}$, and $upd^{\sigma, \rho}$ is the empty sequence ϵ if ρ is green. Here ϵ is the empty sequence of operations (i.e., do nothing to the counter).
- The acceptance condition Φ states that $\limsup_i c(\rho, i) < \infty$ (i.e., counter c must be bounded) and some state $q \in Q \backslash \{(1, s) \mid s \in S^{\multimap}\}$ is visited infinitely often.

Lemma 7. *An edge (s_1, σ, s_2) of $P^{\multimap}$ is:* (i) *red iff it does not appear on any pseudo-cycle of $\mathcal{P}^{\multimap}$;* (ii) *blue iff it appears on a pseudo-cycle of $\mathcal{P}^{\multimap}$ with no broadcasts, but not on any that contain broadcasts;* (iii) *green iff it appears on a pseudo-cycle of $\mathcal{P}^{\multimap}$ with no broadcasts, that is part of a bigger pseudo-cycle with broadcasts;* (iv) *orange iff it appears on a pseudo-cycle C of $\mathcal{P}^{\multimap}$ that has broadcasts, but not on any without broadcasts.*

The following lemma states that we can assume that pseudo-cycles mentioned in Lemma 7 (that have broadcasts) are of a specific form.

Lemma 8. *An edge e appears on a pseudo-cycle D in $\mathcal{P}^{\multimap}$, which contains broadcasts, iff it appears on a pseudo cycle C of $\mathcal{P}^{\multimap}$ containing exactly r broadcast transitions and with all processes starting in the component P_n, where n, r are the prefix length and period of $P^{\multimap}$, respectively. Furthermore, C preserves any nested pseudo-cycles of D that contain no broadcasts.*

Theorem 2. *The language recognized by $\mathcal{A}_{\mathcal{P}}^{\infty}$ is exactly $\mathrm{EXEC}_{\mathcal{P}}^{\infty}$.*

Proof (sketch). The fact that every word in $\mathrm{EXEC}_{\mathcal{P}}^{\infty}$ is accepted by $\mathcal{A}_{\mathcal{P}}^{\infty}$ follows in a straightforward way from its construction. For the reverse direction, given $\alpha \in \mathrm{EXEC}_{\mathcal{P}}^{\infty}$ with an accepting run Ω in $\mathcal{A}_{\mathcal{P}}^{\infty}$, we need to construct a run π in $\mathcal{P}$ whose projection on process 1 is α. We consider the interesting case that α has infinitely many broadcasts (and thus finitely many red and blue edges). The challenging part is how to make process 1 trace the suffix β of α containing only green and orange edges. Since Ω is accepting, counter C_2 is bounded on Ω. Hence, there is a bound $\mathfrak{m}$ on the number of orange edges in β between any r broadcasts, where r is the period of $P^{-\circ}$.

For every green (resp. orange) edge e of P that appears on β, by Lemmas 7, 8, there is a pseudo-cycle C_e with r broadcasts on which e appears. Furthermore, if e is green it actually appears on an inner pseudo-cycle of C_e without broadcasts. Let E_{green} (resp. E_{orange}) be the set of green (resp. orange) edges that appear infinitely often on α. By taking exactly enough processes to assign them to one copy of C_e for every $e \in E_{\mathsf{green}}$, and $\mathfrak{m}$ copies of C_e for every $e \in E_{\mathsf{orange}}$, and composing them using Lemma 1 we can simulate all these copies of these pseudo-cycles in one pseudo-cycle D also with r broadcasts. By Lemma 2, we can pump this pseudo-cycle forever. Furthermore, between broadcasts we have freedom on how to interleave the simulations. We make process 1 trace β by making it successively swap places with the right process in the group simulating a copy of the cycle C_e where e is the next edge on β to be traced (just when the group is ready to use that edge). The key observation is that once a group is used by process 1 there are two options. If it is a group corresponding to a green edge then we can make the group (after 1 leaves it) traverse the inner pseudo-cycle (the one without broadcasts) thus making it ready to serve process 1 again. If the group corresponds to an orange edge e, then it will only be reusable when the whole pseudo-cycle C_e completes (since there is no inner pseudo-cycle to use), i.e., after r broadcasts. However, since there are $\mathfrak{m}$ groups for each such edge, and $\mathfrak{m}$ bounds from above the number of orange edges that need to be taken by process 1 between r broadcasts. □

As we show (Section 4.1, Theorem 4), the problem of determining the type (blue, green, orange, or red) of an edge in $P^{-\circ}$ is decidable, hence, we conclude this section by stating our main theorem (the proof is now immediate).

Theorem 3. *The PMCP (of RB-systems or discrete timed networks) for BW- or SW-specifications or complements of specifications given by BSW, is decidable.*

4.1 Deciding Edge Types

Theorem 4. *Given a process template $P^{-\circ}$, the problem of determining the type (blue, green, orange, red) of an edge e in $P^{-\circ}$ is decidable.*

A key observation for proving Theorem 4 is that by Lemma 7, the type of an edge can be decided by looking for witnessing pseudo-cycles C in $\mathcal{P}^{-\circ}$. Indeed,

a witness can determine if an edge is green or not. If not, another witness can determine if it is orange or not, and the last witness can separate the blue from the red. We will show an algorithm that given an edge that is not green tells us if it is orange or not. The algorithm can be modified to check for the other types of witnesses without much difficulty.

By Lemma 8, we can assume that the pseudo-cycle C we are looking for has very specific structure. Our algorithm uses linear programming, in a novel and interesting way, to detect the existence of such a pseudo-cycle C.

Counter Representation. Given a process template $P = \langle S, I, R, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\} \rangle$, let $d = |S|$, and fix once and for all some ordering $s_1, s_2, \ldots, s_d$ of the states in S. We associate with every configuration f in $\mathcal{P}$ a vector $f^\sharp := (|f^{-1}(s_1)|, \ldots, |f^{-1}(s_d)|) \in \mathbb{N}_0^d$, called the *counter representation* of f. We also associate with every transition $t = (f, \sigma, g)$ the vector $t^\sharp := g^\sharp - f^\sharp$ representing the change in the number of processes in each state. If t is a rendezvous transition then $g^\sharp - f^\sharp$ is completely determined by the action $\mathsf{a} \in \Sigma_{\mathsf{actn}}$ taken in σ. Indeed, if $\sigma = ((i_1, \mathsf{a}_1), \ldots, (i_k, \mathsf{a}_k))$ then $g^\sharp - f^\sharp = \mathsf{a}^\sharp$, where $\mathsf{a}^\sharp \in \mathbb{N}_0^d$ is the vector defined by letting $\mathsf{a}^\sharp(s) := |\{j \in [k] \mid \mathsf{dst}(\mathsf{edge}(a_j)) = s\}| - |\{j \in [k] \mid \mathsf{src}(\mathsf{edge}(a_j)) = s\}|$ for every $s \in S$.

Given $u \in \mathbb{Q}^d$, and a sequence of vectors $\varrho = \varrho_1 \ldots \varrho_m$ in $\mathbb{Q}^d$, the pair $\rho = (u, \varrho)$ is called a path from u to $v = u + \Sigma_{i=1}^m \varrho_i$. We write ρ_j for the vector $u + \Sigma_{i=1}^j \varrho_i$, for every $0 \leq j \leq m$. The path ρ is *legal* if $\rho_j \in \mathbb{Q}_{\geq 0}^d$ for every $0 \leq j \leq m$, i.e., if no coordinate goes negative at any point. Given a finite path $\pi_1 \ldots \pi_m$ in $\mathcal{P}$, we call the path $\pi^\sharp := (\mathsf{src}(\pi_1)^\sharp, \pi_1^\sharp \ldots \pi_m^\sharp)$ in $\mathbb{Q}^d$ its *counter representation*. Observe that $\pi^\sharp$ is always a legal path.

Rational Relaxation of VASs. Vector Addition Systems (VASs) or equivalently Petri nets are one of the most popular formal methods for the representation and the analysis of parallel processes [10]. Unfortunately, RB-systems **cannot** be modelled by VASs since a transition in a VAS only moves a constant number of processes, whereas a broadcast in an RB-system may move any number of processes. On the other hand, R-System can be modelled by VASs, and we do use this fact to analyze the behaviour of the counter representation between broadcasts. Moreover, we note that integer linear programming is a natural fit for describing paths and cycles in the counter representation. However, in order to apply linear programming to RB-systems we have to overcome two intertwined obstacles: *(i)* not every path in the counter representation induces a path in $\mathcal{P}$, and *(ii)* since we have no bound on the length of the pseudo-cycle C we cannot have variables describing each configuration on it, and we need to aggregate information. These obstacles are aggravated by the presence of broadcasts. Another difficulty of applying linear programming to RB-systems arises from the fact that the question of reachability in an RB-system with two (symmetric) broadcast actions and a controller is undecidable (which can be obtained by modifying a result in [9] concerning asymmetric broadcast).

The solution we propose to this problem, which we found to be surprisingly powerful, is to use linear programming but look for a solution in *rational* numbers and not in integers. Thus, we introduce the notion of the *rational relaxation* of a

VAS, obtained by allowing any non-negative rational multiple of configurations and transitions of the original VAS. Since our linear programs use homogeneous systems of equations, multiplying a rational solution by a large enough number would yield another solution in integers. Thus the scaling property obtained a consequence of rational relaxation precludes the possibility of specifying a single controller! Thinking of the counter representation as vectors of rational numbers also allows us to use geometric reasoning to solve the two problems (i), (ii) described above. Essentially, by cutting transitions to smaller pieces (which cannot be done at will to integer vectors) and rearranging the pieces, we can transform a description of a path in an aggregated form, as it comes out of the linear program, into one which is legal and can be turned into a path in $\mathcal{P}$. We strongly believe that these techniques can be fruitfully used in other circumstances concerning counter-representations, and similar objects (such as vector addition systems and Petri nets).

Due to lack of space, the description of the linear programs we use, as well as the geometric machinery we develop will be published in an extended version.

References

1. Abdulla, P.A., Deneux, J., Mahata, P.: Multi-clock timed networks. In: Ganzinger, H. (ed.) LICS. pp. 345–354 (July 2004)
2. Abdulla, P.A., Jonsson, B.: Model checking of systems with many identical timed processes. TCS 290(1), 241–264 (2003)
3. Alur, R.: Timed automata. In: CAV. pp. 8–22. Springer (1999)
4. Aminof, B., Jacobs, S., Khalimov, A., Rubin, S.: Parameterized model checking of token-passing systems. In: VMCAI. pp. 262–281. Springer (2014)
5. Aminof, B., Kotek, T., Rubin, S., Spegni, F., Veith, H.: Parameterized model checking of rendezvous systems. In: CONCUR, pp. 109–124. Springer (2014)
6. Bojanczyk, M.: Beyond ω-regular languages. In: STACS 2010. pp. 11–16 (2010)
7. Chevallier, R., Encrenaz-Tiphene, E., Fribourg, L., Xu, W.: Timed verification of the generic architecture of a memory circuit using parametric timed automata. Formal Methods in System Design 34(1), 59–81 (2009)
8. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: CONCUR. LNCS, vol. 6269, pp. 313–327 (2010)
9. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS. pp. 352–359 (1999)
10. Esparza, J., Nielsen, M.: Decidability issues for petri nets - a survey. Bulletin of the EATCS 52, 244–262 (1994)
11. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. JACM 39(3), 675–735 (1992)
12. Kouvaros, P., Lomuscio, A.: A cutoff technique for the verification of parameterised interpreted systems with parameterised environments. In: IJCAI 2013 (2013)
13. Spalazzi, L., Spegni, F.: Parameterized model-checking of timed systems with conjunctive guards. In: Verified Software: Theories, Tools and Experiments, pp. 235–251. Springer (2014)
14. Suzuki, I.: Proving properties of a ring of finite-state machines. Inf. Process. Lett. 28(4), 213–214 (1988)
15. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Logics for concurrency, pp. 238–266. Springer (1996)