Logical Methods in Computer Science
Volume 21, Issue 2, 2025, pp. 27:1–27:59
https://lmcs.episciences.org/

Submitted Oct. 09, 2023
Published Jun. 24, 2025

PARAMETERIZED MODEL-CHECKING OF DISCRETE-TIMED NETWORKS AND SYMMETRIC-BROADCAST SYSTEMS

BENJAMIN AMINOF [a,d], SASHA RUBIN [b], FRANCESCO SPEGNI [c], AND FLORIAN ZULEGER [a]

[a] Technical University of Vienna, Austria

[b] University of Sydney, Australia

[c] Università Politecnica delle Marche, Ancona, Italy

[d] Università degli Studi di Roma La Sapienza, Italy

Abstract. We study the complexity of the model-checking problem for parameterized discrete-timed systems with arbitrarily many anonymous and identical contributors, with and without a distinguished "controller", and communicating via asynchronous rendezvous. Our work extends the seminal work from German and Sistla on untimed systems by adding discrete-time clocks to processes.

For the case without a controller, we show that the systems can be efficiently simulated — and vice versa — by systems of untimed processes that communicate via rendezvous and symmetric broadcast, which we call "RB-systems". Symmetric broadcast is a novel communication primitive that allows all processes to synchronize at once; however, it does not distinguish between sending and receiving processes.

We show that the parameterized model-checking problem for safety specifications is PSPACE-complete, and for liveness specifications it is decidable in EXPTIME. The latter result is proved using automata theory, rational linear programming, and geometric reasoning for solving certain reachability questions in a new variant of vector addition systems called "vector rendezvous systems". We believe these proof techniques are of independent interest and will be useful in solving related problems.

For the case with a controller, we show that the parameterized model-checking problems for RB-systems and systems with asymmetric broadcast as a primitive are inter-reducible. This allows us to prove that for discrete timed-networks with a controller the parameterized model-checking problem is undecidable for liveness specifications.

Our work exploits the intimate connection between parameterized discrete-timed systems and systems of processes communicating via broadcast, providing a rare and surprising decidability result for liveness properties of parameterized timed-systems, as well as extend work from untimed systems to timed systems.

Key words and phrases: Parameterized systems, timed-systems, broadcast communication, decidability, formal languages.

LOGICAL METHODS IN COMPUTER SCIENCE

DOI:10.46298/LMCS-21(2:27)2025

1. Introduction

We systematically study the complexity of the model-checking problem for parameterized discrete-timed systems that communicate via synchronous rendezvous. These systems consist of arbitrarily many anonymous and identical contributors, with and without a distinguished "controller" process. The parameterized model-checking problem asks whether a given specification holds no matter the number of identical contributors. This is in contrast to traditional model-checking that considers a fixed number of contributors.

Our model subsumes the classic case of untimed systems [GS92] — processes are finite-state programs with (discrete-time) clocks that guard transitions. Timed processes can be used to model more realistic circuits and protocols than untimed processes [Alu99, CETFX09].

We study the computational complexity of the parameterized model-checking problem (PMCP) for safety and liveness specifications.[1] Safety properties are specified by formulas of linear-temporal logic over finite traces (LTLf) and nondeterministic finite word automata (NFW), and liveness properties are specified by formulas of linear-temporal logic (LTL) and nondeterministic Büchi automata (NBW).

We show that without a controller safety is PSPACE-complete, while liveness is in EXPTIME; and with a controller safety is decidable, while liveness is undecidable. In more detail:

(1) For systems without a controller, we prove that the PMCP for safety specifications is PSPACE-complete — in fact, PSPACE-hardness even holds for a fixed specification (known as program complexity) and for a fixed program (known as specification complexity).
(2) For systems without a controller, we prove that the PMCP for liveness specifications can be solved in EXPTIME. This is a rare decidability result for liveness properties of any non-trivial model of parameterized timed systems. The algorithms presented make use of interesting and intricate combination of automata theory, rational linear programming, and geometric reasoning for solving certain reachability questions in a new variant of vector addition systems called 'vector rendezvous systems'. We believe these techniques are of independent interest and will be useful in solving related problems.
(3) For systems with a controller, we prove that the PMCP for liveness specifications is undecidable. This follows from a new reduction between timed-systems with a controller and systems with asymmetric broadcast, and the known undecidability of the PMCP of latter for liveness properties. The novel reduction also allows us to recover the known result that PMCP is decidable for discrete-time systems with a controller and safety specifications.

Although this doesn't completely close the picture (the complexity without a controller for liveness properties is PSPACE-hard and in EXPTIME), we remark that the parameterized verification problem for liveness properties is notoriously hard to analyze: apart from a single simple cutoff result which deals with processes communicating using conjunctive and disjunctive guards [SS20, SS14], no decidability result for liveness specifications of timed systems was known before this work.

To solve the PMCP problem for these systems we introduce *rendezvous-broadcast systems (RB-systems)* — systems of finite-state processes communicating via *rendezvous* and *symmetric broadcast*. Unlike asymmetric broadcast which can distinguish between the sender and the receivers, with symmetric broadcast there is no designated sender, and thus it can

[1] We follow Esparza et. al. [EFM99] and refer to sets of finite executions as *safety* properties, and sets of infinite executions as *liveness* properties.

naturally model the passage of discrete time, i.e., every symmetric broadcast can be thought of as a tick of the discrete-time clocks.

We show that RB-systems and timed-networks with the same number of processes can efficiently simulate each other. Thus, in particular, the PMCP of RB-systems and timed networks are polynomial-time inter-reducible. Furthermore, we show that for the case with a controller, RB-systems (and thus timed-networks) are polynomial-time inter-reducible to systems with asymmetric broadcasts. We remark that this equivalence does not hold for the case without a controller (indeed, we show that without a controller PMCP for liveness specifications is decidable, whereas it is known to be undecidable for systems with asymmetric broadcast [EFM99]). We thus consider the introduction of the notion of a symmetric broadcast to be an interesting communication primitive in itself. We then study the PMCP for RB-systems and in fact establish the itemized results above for RB-systems.

This work extends the results and provides detailed versions of proofs and statements that already appeared in the conference proceedings version [ARZS15].

1.1. **Techniques.** The bulk of the work concerns the case without a controller, in which we first establish the decidability of the PMCP for safety properties and prove this problem to be PSPACE-complete (already for a fixed specification). The decidability of the PMCP for safety properties of timed networks has already been known [ADM04]; however, it was obtained using well-structured transition systems and only gives a *non-elementary* upper bound, which we improve to PSPACE. We obtain the result for safety properties by constructing a reachability-unwinding of the states of the processes of the parameterized system, where we compute precisely those states the system can be in after exactly n broadcasts; we show that the reachability-unwinding has a lasso-shape and can be constructed in PSPACE, which allows us to obtain the upper complexity bound. We provide a matching lower-bound by reducing the termination problem of Boolean programs to non-reachability in RB-systems.

We then prove an EXPTIME upper-bound for the PMCP for liveness properties. This result is considerably more challenging than the upper-bound for safety properties. One source of difficulty is the need to be able to tell whether some rendezvous transition can be executed a bounded (as opposed to unbounded) number of times between two broadcasts — a property which is not ω-regular. In order to deal with this issue we work with B-automata [Boj10], which generalize Büchi-automata by equipping them with counters.

A key step in constructing a B-automaton that recognizes the computations of the system that satisfy a liveness specification requires establishing the existence (or lack thereof) of certain cycles in the runs of the parameterized system. Alas, the intricate interaction between broadcasts and rendezvous transitions makes this problem very complicated. In particular, known classical results concerning pairwise rendezvous without broadcast [GS92] do not extend to our case. We solve this problem in two steps: we first obtain a precise characterization (in terms of a set of linear equations) for reachability of configurations between two broadcasts; we then use this characterization in an iterative procedure for establishing the existence of cycles with broadcasts. To obtain the characterization for reachability mentioned above we introduce vector rendezvous systems (VRS) and their continuous relaxation, called continuous vector rendezvous systems (CVRS). These systems are counter abstractions — which view configurations as vectors of counters that only store the number of processes at every process state, but not the identity of the processes — and constitute a new variant of the classical notion of vector addition systems [HP79].

1.2. **Related Work.** In this work we parameterize by the number of processes (other choices are possible, e.g., by the spatial environment in which mobile-agents move [AMRZ22]). Our model assumes arbitrarily many finite-state anonymous identical processes, possibly with a distinguished controller. Finite-state programs are commonly used to model processes, especially in the parameterized setting [AST18, BJK+15, CTV08, GS92]. The parameterized model-checking problem (PMCP) has been studied for large-scale distributed systems such as swarm robotics [LP22, KL16], hardware design [McM01], and multi-threaded programs [KKW14].

The PMCP easily becomes undecidable for many types of communication primitives because of the unbounded number of processes (even for safety properties and untimed processes) [Suz88]. Techniques for solving the general case include identifying decidable/tractable subcases (as we do) and designing algorithms that are not guaranteed to terminate, e.g., with acceleration and approximation techniques [AST18, ZP04]. The border between decidability and undecidability for various models is surveyed in [BJK+15], including token-passing systems [AR16, AJKR14], rendezvous and broadcast [GS92, EFM99, AKR+18], guarded protocols [JS18], ad hoc networks [DSZ10].

The seminal work [GS92] shows that the PMCP of (untimed) systems that communicate via rendezvous is in PTIME without a controller, and is EXPSPACE-complete with a controller. We now compare our work against others that explored the PMCP for timed processes (and timed or untimed specifications).

In [BF13], Bertrand and Fournier consider the case of dynamic networks of timed Markov decision processes (TMDPs), a model for agents mixing probabilistic and timed behavior, whose transitions can be guarded by simple conditions on a global real-valued clock variable and through broadcast messages (a broadcaster is distinguished from the receivers) that can force all processes to transition in one computational step. They explore the decidability of several variants of the parameterized probabilistic reachability problem in networks of TMDPs. Interestingly, they observe that in some settings the problem is undecidable if the number of processes is fixed but unknown, while it becomes decidable (but not primitive recursive) by allowing processes to join and leave the network with given probabilities along the executions of the system. Their undecidability results are based on the possibility, using message broadcasting, of distinguishing a process that acts as controller of the network.

In [AJ03] Abdulla and Jonsson prove that model checking safety properties for timed networks with a controller process is decidable, provided that each process has at most one (real-valued) clock variable. They assume several processes can take synchronous transitions at once by using a rendezvous primitive and a distinguished process can act as controller of the network. The decidability carries over to discrete timed networks with one clock variable per process, and in case all processes are equal, i.e. the controller process is just another copy of the user processes. Our work proves decidability of safety and liveness properties independently from the number of clock variables in the network, provided that clocks range over a discrete domain and there is no controller process in the network.

In [ADM04] Abdulla et al. extends the decidability result of [AJ03] to timed networks with rendezvous and a controller process, assuming each process has any finite number of discrete clocks. They prove decidability by exhibiting a non-elementary complexity upper bound for the problem, and of course this also proves decidability of the same problem for timed network in the absence of a controller process. In our setting, without a controller process, we are able to prove a much smaller upper bound for the complexity of the model checking problem restricted to safety properties, as well as the decidability of liveness

properties. In case there is a controller process, our results show that the PMCP for liveness properties for timed networks is undecidable.

In [ADR+16] Abdulla et al. consider the PMCP for a model of timed processes communicating through Ad Hoc Wireless Networks. In this model, an arbitrary number of processes use either real- or discrete- valued clocks and can connect among themselves in topologies that are defined by some given family of graphs such as bounded path graphs or cliques. Given a family of graphs, each timed process can communicate with its direct neighbors using broadcast or rendezvous messages. In the context of timed networks using rendezvous on a clique graph topology, which is the setting closest to our work, they focus on rendezvous communications and rephrase the decidability results already presented in [AJ03, ADM04].

In [Ise17] Isenberg provides techniques for finding invariants for safety properties and timed-networks consisting of processes with continuous clocks, a controller, shared global variables and broadcast communication. In contrast, our decision procedures are based on automata theory.

In [AAC+18], Abdulla et al. study the PMCP for reachability specifications for Timed Petri Nets. There the authors prove that the problem is PSPACE-complete provided that each process carry only one clock variable. Interestingly, this is the same complexity we prove for the discrete case, thus the extension of the problem to the continuous time setting (under the limitation of one clock variable per process) falls in the same complexity class as the discrete time setting. In this work we make a step further by providing an upper bound to the complexity of the PMCP for liveness specifications in the discrete time setting.

In [SS20, SS14] Spalazzi and Spegni study the parameterized model-checking problem of Metric Interval Temporal Logic formulae against networks of conjunctive or disjunctive timed automata of arbitrary size. They prove that in case of timed networks with either all conjunctive or all disjunctive Boolean guards and a controller process, a cutoff exist allowing to reduce controlled timed networks of arbitrary size to timed networks of some known size, provided that process locations have no time invariants forcing progress. This implies that PMCP is decidable under such conditions. In contrast, in our setting time is discrete, there is no controller process, communication is by rendezvous, and specifications are qualitative (i.e. LTL).

Andrè et al. [AEJK23] prove that a cutoff for timed networks with all disjunctive Boolean guards does exist in presence of clock invariants in the case of processes with a single clock variable and suitable conditions ensuring that locations appearing in the clock invariants themselves are visited infinitely often.

Finally, we remark that simulations between parameterized systems with different communication primitives, including asymmetric broadcast and rendezvous, is systematically studied in [ARZ15]. We also contribute to that line of work, involving the newly introduced symmetric-broadcast primitive. Our work establishes for the first time an intimate two-way connection between discrete-timed systems and systems communicating via broadcast: symmetric broadcast when there is no controller, and asymmetric broadcast when there is a controller. We are certain that this intimate connection will prove useful in transferring results between these two types of systems, and discovering new results also if one considers other communication primitives than rendezvous.

2. Definitions and Preliminaries

For the sake of self-consistency, let us now recall various notions from automata theory that will be used along this work.

Notation. Let $\mathbb{N}_{>0}$ denote the set of positive integers, let $\mathbb{N} = \mathbb{N}_{>0} \cup \{0\}$, let $\mathbb{Q}$ denote the set of rational numbers, $\mathbb{Q}_{>0}$ the set of positive rational numbers, and $\mathbb{Q}_{\geq 0}$ the non-negative ones. Let $[n, m]$, for $n < m \in \mathbb{N}_{>0}$, denote the set $\{n, n+1, \ldots, m\}$, and if $m = \infty$ then $[n, m] = \{n, n+1, \ldots\}$. Let $[n]$ denote the set $[1, n]$. Finally, for $m \leq n \in \mathbb{N}_{>0}$, we call $\mu : [m] \to 2^{[n]}$, a *partition* of $[n]$ if $i \neq j$ implies $\mu(i) \cap \mu(j) = \emptyset$, and $[n] = \cup_{i \in [m]} \mu(i)$. For an *alphabet* Σ we denote by Σ^* (resp. Σ^ω) the set of all finite (resp. infinite) *words* over Σ. The *concatenation* of two words u and w is written uw or $u \cdot w$. The *length* of a word u is denoted by $|u|$, and if u is infinite then we write $|u| = \infty$.

2.1. Transition Systems.

A *labeled transition system (LTS)* is a tuple

$$L = \langle AP, \Sigma, S, I, R, \lambda \rangle$$

where

- AP is a finite set of *atomic propositions* (also called *state labels*),
- Σ is an alphabet of *edge-labels*,
- S is a set of *states* (in the following we assume that $S \subseteq \mathbb{N}_{>0}$),
- $I \subseteq S$ is a set of *initial states*,
- $R \subseteq S \times \Sigma \times S$ is an *edge relation*,
- and $\lambda \subseteq S \times AP$ is a *labeling relation* that associates with each state the atomic propositions that hold in it. We will often use functional notation and write $\lambda(s)$ for the set of atoms p such that $(s, p) \in \lambda$.

In case all components of L are finite, we say that L is a *finite LTS*; and otherwise we say that it is an *infinite LTS*. An edge $e = (s, a, s') \in R$, is also called a *transition*, and may be written $s \xrightarrow{a} s'$. The element s is called the *source* (denoted $\mathsf{src}(e)$) of e, and s' is called its *destination* (denoted $\mathsf{dst}(e)$), and a is called the *label* of e. Given $\sigma \in \Sigma$, and a state $s \in S$, we say that σ is *enabled* in s if there is some $s' \in S$ such that $s \xrightarrow{\sigma} s'$. A *path* π is a (finite or infinite) sequence $e_1 e_2 \ldots$ of transitions such that for every $1 \leq i < |\pi|$ we have that $\mathsf{dst}(e_i) = \mathsf{src}(e_{i+1})$, where $|\pi| \in \mathbb{N}_{>0} \cup \{\infty\}$ is the *length* of π. We extend the notations $\mathsf{src}(\pi)$ and $\mathsf{dst}(\pi)$ to paths (the latter only for finite paths) in the natural way. Extend λ to paths as follows: if $\pi = e_1 e_2 \cdots e_k$ is finite then $\lambda(\pi) = \lambda(\mathsf{src}(e_1))\lambda(\mathsf{src}(e_2)) \cdots \lambda(\mathsf{src}(e_{k-1}))\lambda(\mathsf{src}(e_k))\lambda(\mathsf{dst}(e_k))$, and if $\pi = e_1 e_2 \cdots$ is infinite then $\lambda(\pi) = \lambda(\mathsf{src}(e_1))\lambda(\mathsf{src}(e_2)) \cdots$. A *run* is a path whose source is an initial state. The set of runs of an LTS L is written *runs*(L). A state $s \in S$ is *reachable* if it is the destination of some run. The *size* of a finite LTS is defined to be the sum of the number of states and number of transitions.

Let $L = \langle AP, \Sigma, S, I, R, \lambda \rangle$ and $L' = \langle AP, \Sigma, S', I', R', \lambda' \rangle$ be two LTSs over the same set of atomic propositions AP and the same set of edge-labels Σ. We now define a few notions of equivalence relating such LTSs. A relation $M \subseteq S \times S'$ is a *simulation* if (i) for every $q \in I$ there is $q' \in I'$ such that $(q, q') \in M$, (ii) $(q, q') \in M$ implies $\lambda(q) = \lambda'(q')$ and for every $(q, \sigma, r) \in R$ there exists r' with $(q', \sigma, r') \in R'$ such that $(r, r') \in M$. In this case we say that L' *simulates* L. Say that M is a *bisimulation* if M is a simulation and $\{(q', q) : (q, q') \in M\} \subseteq Q' \times Q$ is a simulation. We say that runs π, π', of L and L' respectively, of the same length are *equi-labeled* if for every $i < |\pi|$, if $\pi_i = (s, \sigma, t)$ and

$\pi'_i = (s', \sigma', t')$ we have that $\lambda(s) = \lambda'(s')$, $\lambda(t) = \lambda'(t')$, and $\sigma = \sigma'$. It follows immediately from the definitions that if L' simulates L then for every run in L there exists an equi-labeled run in L'.

We will use the following operations: Let AP be a set of atomic propositions. Given a proposition $a \in AP$ and a (finite or infinite) sequence $\xi \in (2^{AP})^* \cup (2^{AP})^\omega$, we denote by $(\xi)_a$ the subsequence of ξ that consists of all sets that contain a. Given a subset $AP' \subseteq AP$ and a (finite or infinite) sequence $\xi \in (2^{AP})^* \cup (2^{AP})^\omega$, we denote by $\xi|_{AP'}$ the sequence that we obtain from ξ by intersecting every set with AP'.

2.2. **Automata.** We will use nondeterministic automata with three types of acceptance conditions, i.e., ordinary reachability acceptance (on finite input words), Büchi acceptance (on infinite words), and a boundedness condition on a single counter (on infinite words). Since automata are like LTSs (except that they include an acceptance condition, and exclude the state labeling function), we will use LTS terminology and notation that is independent of the labeling, e.g., source, destination, path and run. We remark that inputs to the automata will be edge-labeling of paths in certain LTSs, and thus the input alphabet for automata is also denoted Σ.

A *nondeterministic finite word automaton (NFW)* is a tuple

$$\mathcal{A} = \langle \Sigma, S, I, R, F \rangle$$

where

- Σ is the *input alphabet*,
- S is the finite set of *states*,
- $I \subseteq S$ are the *initial states*,
- $R \subseteq S \times \Sigma \times S$ is the *transition relation*, and
- $F \subseteq S$ are the *final states*.

Given a finite word $\alpha = \alpha_1 \alpha_2 \cdots \alpha_k$ over the alphabet Σ, we say that $\rho = \rho_1 \rho_2 \dots \rho_k$ is a *run of $\mathcal{A}$ over α* if, for all $i \in [k]$, the label of the transition ρ_i is α_i. The run ρ is *accepting* if $\mathsf{dst}(\rho_k) \in F$. A word is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ over it. The *language* of $\mathcal{A}$ is the set of words that it accepts.

A *nondeterministic Büchi word automaton (NBW)* is a tuple $\mathcal{A} = \langle \Sigma, S, I, R, G \rangle$, which is like an NFW except that F is replaced by a *Büchi set* G. Unlike NFW which run over finite words, an NBW runs over infinite words. Hence, given an infinite word $\alpha = \alpha_1 \alpha_2 \cdots$ over the alphabet Σ, we say that $\rho = \rho_1 \rho_2 \dots$ is a *run of $\mathcal{A}$ over α* if, for all $i \in \mathbb{N}_{>0}$, the label of the transition ρ_i is α_i. The run ρ induces a set $inf(\rho)$ consisting of those states $q \in S$ such that $q = \mathsf{src}(\rho_i)$ for infinitely many i. The run ρ is *accepting* if $inf(\rho) \cap G \neq \emptyset$. The definition when a word is *accepted*, and of the *language* of $\mathcal{A}$, are as for NFW.

An *NBW with one counter*, or *B-automaton* for short, is a tuple

$$\langle \Sigma, S, I, R, G, cc \rangle$$

which is like an NBW except that it has an additional *counter command function* $cc : R \to \{\mathsf{inc}, \mathsf{reset}, \mathsf{skip}\}$ which associates with each transition a counter-update operation. An infinite run $\rho = \rho_1 \rho_2 \cdots$ of a B-automaton induces a set $ctr(\rho) = \{c_i : i \in \mathbb{N}_{>0}\}$, where $c_1 = 0$ and

$$c_{i+1} = \begin{cases} c_i & \text{if } cc(\rho_i) = \mathsf{skip} \\ c_i + 1 & \text{if } cc(\rho_i) = \mathsf{inc} \\ 0 & \text{if } cc(\rho_i) = \mathsf{reset}. \end{cases}$$

The run ρ is *accepting* if it satisfies the Büchi condition and its counter values are bounded. I.e., if $inf(\rho) \cap G \neq \emptyset$ and $\exists n \in \mathbb{N}_{>0}$ s.t. $c < n$ for all $c \in ctr(\rho)$.

If $G = S$ (i.e., if there is effectively no Büchi acceptance condition), then we say that the Büchi set is *trivial*.

B-automata were defined in [Boj10], and in the general case may have multiple counters, some of which should be bounded and some of which should be unbounded. Since one can easily simulate a Büchi acceptance condition with a single counter (see [Boj10]), our definition of a B-automaton given above is a special case of the B-automata of [Boj10] with two counters. The proof of Lemma 2.1 below, which also applies to these general multi-counter automata, was communicated to us by Nathanaël Fijalkow (as far as we know it is a "folk theorem" for which we could not find a clear statement or proof in the literature).

Lemma 2.1. *Deciding whether the language of a B-automaton is not empty can be solved in* PTIME.

Proof. We reduce the problem to the emptiness problem for Streett automata [2], which is in PTIME [EL87].

Given a B-automaton $\mathcal{A} = \langle \Sigma, S, I, R, G, cc \rangle$, build a Streett automaton $\mathcal{A}'$ whose transition relation is like that of $\mathcal{A}$ except that it also stores the most recent counter command in the state, and whose acceptance condition encodes the following properties: 'infinitely often see a state in G' and 'infinitely many increments implies infinitely many resets'. Formally, $\mathcal{A}'$ has states $S \times \{\mathsf{skip}, \mathsf{inc}, \mathsf{reset}\}$; initial states $S \times \{\mathsf{reset}\}$; transitions of the form $((s, c), \alpha, (s', c'))$ where $(s, \alpha, s') \in R$ and $cc(s, \alpha, s') = c'$; and the acceptance condition containing the two pairs (S, G) and $(S \times \{\mathsf{inc}\}, S \times \{\mathsf{reset}\})$.

Then, the language of $\mathcal{A}$ is non-empty if and only if the language of $\mathcal{A}'$ is non-empty. To see this, note that accepting runs in $\mathcal{A}$ induce accepting runs in $\mathcal{A}'$, since a run with infinitely many "increments" that also has a bound on the counter must have infinitely many "resets". On the other hand, an accepting run ρ_1 of $\mathcal{A}'$ can be transformed, by "pumping out" loops, into another accepting run ρ_2 of $\mathcal{A}'$ in which the distance between two successive reset transitions is bounded (one only needs to ensure that in each infix starting and ending in a reset, if there is a Büchi state in the infix, then there is still one after pumping out). Thus, the counter of ρ_2 is bounded, and so is also an accepting run of $\mathcal{A}$.[3] □

2.3. **Linear Temporal Logic.** For a set AP of atomic propositions, *formulas of LTL over* AP are defined by the following BNF (where $p \in AP$):

$$\varphi ::= p \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi$$

We use the usual abbreviations, $\varphi \to \varphi' = \neg\varphi \vee \varphi'$, $\mathtt{true} = p \vee \neg p$, $\mathsf{F}\,\varphi = \mathtt{true}\,\mathsf{U}\,\varphi$ (read "eventually φ"), $\mathsf{G}\,\varphi = \neg\,\mathsf{F}\,\neg\varphi$ (read "always φ"). The *size* $|\varphi|$ of a formula φ is the number of symbols in it. A *trace* τ is an infinite sequence over the alphabet $\Sigma = 2^{AP}$, an infinite sequence of valuations of the atoms. For $n \geq 0$, write τ_n for the valuation at position n; so, $\tau = \tau_0\tau_1\tau_2 \cdots$ Given a trace τ, an integer n, and an LTL formula φ, the satisfaction relation $(\tau, n) \models \varphi$, stating that φ holds at step n of the sequence τ, is defined as follows:

[2] We remind the reader that a Streett automaton is like an NBW except that its acceptance condition is not a single Büchi set, but a family of pairs of sets $\{(B_1, G_1), (B_2, G_2), \ldots, (B_k, G_k)\}$, and a run ρ is accepting if for all $i \in [k]$ we have that $inf(\rho) \cap G_i \neq \emptyset$ implies $inf(\rho) \cap B_i \neq \emptyset$.

[3] Observe that the Streett automaton does not, in general, accept the same language as the B-automaton. Indeed the latter's language may not even be ω-regular.

- $(\tau, n) \models p$ iff $p \in \tau_n$;
- $(\tau, n) \models \varphi_1 \vee \varphi_2$ iff $(\tau, n) \models \varphi_1$ or $(\tau, n) \models \varphi_2$;
- $(\tau, n) \models \neg\varphi$ iff it is not the case that $(\tau, n) \models \varphi$;
- $(\tau, n) \models \mathsf{X}\,\varphi$ iff $n + 1 < |\tau|$ and $(\tau, n + 1) \models \varphi$;
- $(\tau, n) \models \varphi_1 \,\mathsf{U}\, \varphi_2$ iff $(\tau, m) \models \varphi_2$ for some $n \leq m < |\tau|$, and $(\tau, j) \models \varphi_1$ for all $n \leq j < m$.

Write $\tau \models \varphi$ if $(\tau, 0) \models \varphi$, read τ *satisfies* φ.

We consider the variant LTLf known as "LTL over finite traces" [BK00, BM06, DGV13]. It has the same syntax and semantics as LTL except that τ is a finite sequence. Observe that the satisfaction of X and U on finite traces is defined "pessimistically", i.e., a trace cannot end before the promised eventuality holds.

The following states that one can convert LTL/LTLf formulas to NBW/NFW with at most an exponential blowup:

Theorem 2.2 [Var95, DGV13]**.** *Let φ be an LTL (resp. LTLf) formula. One can build an NBW (resp. NFW), whose size is at most exponential in $|\varphi|$, accepting exactly the models φ.*

3. Parameterized systems

We first introduce *systems with rendezvous and symmetric broadcast* (or *RB-systems*, for short), a general formalism suitable for describing the parallel composition of $n \in \mathbb{N}_{>0}$ copies of a process *template*. We identify two special cases: Rendezvous systems (or R-systems, for short) and Discrete Timed Systems.

3.1. RB-systems.

An RB-system is a certain LTS which evolves nondeterministically: either a k-wise rendezvous action is taken, i.e., k different processes instantaneously synchronize on some rendezvous action a, or the symmetric broadcast action is taken, i.e., all processes take an edge labeled by $\mathfrak{b}$. Systems without the broadcast action are called R-systems. We will show that RB-systems (strictly) subsume discrete timed networks [ADM04], a formalism allowing to describe parameterized networks of timed processes with discrete value clocks. A discrete timed network also evolves nondeterministically: either a k-wise rendezvous action is taken by k processes of the network, or all the clocks of all the processes advance their value by the same (discrete) amount.

In the rest of the paper the number of processes participating in a rendezvous will be denoted by k, we let Σ_{actn} denote a finite set of *rendezvous actions*, and we call *rendezvous alphabet* the set $\Sigma_{\mathsf{rdz}} = \{a_i \ : \ a \in \Sigma_{\mathsf{actn}}, i \in [1, k]\}$.

Definition 3.1 (**Process Template, RB-Template, R-Template**)**.** A *process template* is a finite LTS $P = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S, I, R, \lambda\rangle$. A process template P is an *RB-template*, if for every state $s \in S$, we have that $\mathfrak{b}$ is enabled in s.We call edges labeled by $\mathfrak{b}$ *broadcast edges*, and the rest *rendezvous edges*. A process template P is an *R-template*, if P does not contain any broadcast edges.

We now define the system P^n consisting of n copies of a given template P:

Definition 3.2 (**RB-System, R-System**)**.** Given an integer $n \in \mathbb{N}_{>0}$ and an RB-Template (resp. R-Template) $P = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S, I, R, \lambda\rangle$, the *RB-system* P^n (resp. *R-system* P^n) is defined as the finite LTS $\langle AP^n, \Sigma^n_{\mathsf{com}}, S^n, I^n, R^n, \lambda^n\rangle$ where:

(1) The set of atomic propositions AP^n is $AP \times [n]$; intuitively, the atom (p, i) denotes the fact that atom p is currently true of process i.
(2) The *communication alphabet* Σ^n_{com} consists of $\mathfrak{b}$ and every tuple of the form $((i_1, a_1), \ldots, (i_k, a_k))$ where $a \in \Sigma_{\mathsf{actn}}$ and $i_1, i_2, \cdots, i_k$ are k different elements in $[n]$; intuitively, the system takes this action means that simultaneously for each $j \in [k]$, process i_j transitions along an a_j edge.
(3) S^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 S^n be the set of functions $f : X \to S$, where $X \subseteq \mathbb{N}_{>0}$ is some set of size n.
(4) The set of *initial configurations* $I^n = \{f \in S^n \mid f(i) \in I \text{ for all } i \in [n]\}$ consists of all configurations which map all processes to initial states of P.
(5) The set of *global transitions* $R^n \subseteq S^n \times \Sigma^n_{\mathsf{com}} \times S^n$ contains transitions $f \xrightarrow{\sigma} g$ where one of the following two conditions hold:
   - *(broadcast)* $\sigma = \mathfrak{b}$, and $f(i) \xrightarrow{\mathfrak{b}} g(i)$ in R, for every $i \in [n]$;
   - *(rendezvous)* $\sigma = ((i_1, a_1), \ldots, (i_k, a_k))$, and $f(i_j) \xrightarrow{a_j} g(i_j)$ in R for every $1 \le j \le k$; and $f(i) = g(i)$ for every $i \notin \{i_1, \ldots, i_k\}$. In this case we say that $a \in \Sigma_{\mathsf{actn}}$ is the *action taken*.
(6) The labeling relation $\lambda^n \subseteq S^n \times AP^n$ consists of the pairs $(f, (p, i))$ such that $(f(i), p) \in \lambda$.

For every transition $t = (f, \sigma, g) \in R^n$, we define the set of *active processes*, denoted by $\mathsf{active}(t)$, as follows:

- if $\sigma = \mathfrak{b}$ define $\mathsf{active}(t) = [n]$,
- if $\sigma = ((i_1, a_1), \ldots, (i_k, a_k))$ define $\mathsf{active}(t) = \{i_1, \ldots, i_k\}$.

Let t be a global transition $f \xrightarrow{\sigma} g$, and let i be a process. We say that i *moved* in t if $i \in \mathsf{active}(t)$. We write $edge_i(t)$ for the edge of P taken by process i in the transition t, i.e.,

- if $\sigma = \mathfrak{b}$ then $edge_i(t)$ denotes $f(i) \xrightarrow{\mathfrak{b}} g(i)$;
- if $\sigma = ((i_1, a_1), \ldots, (i_k, a_k))$ then $edge_i(t)$ denotes $f(i) \xrightarrow{a_j} g(i)$ if $\sigma(j) = (i, a_j)$ for some $j \in [k]$;
- otherwise $edge_i(t) := \perp$.

In case that $edge_i(t) \neq \perp$ we say that $edge_i(t)$ is *taken* in t. Given a run π of P^n and an edge e of P, we say that e *appears* on π if it is taken by some active process on some transition of π.

Given a process template P define the *RB-system* P^∞ as the following LTS:

$$\langle AP^\infty, \Sigma^\infty_{\mathsf{com}}, S^\infty, I^\infty, R^\infty, \lambda^\infty \rangle$$

where $AP^\infty = \cup_{n \in \mathbb{N}_{>0}} AP^n$, $S^\infty = \cup_{n \in \mathbb{N}_{>0}} S^n$, $I^\infty = \cup_{n \in \mathbb{N}_{>0}} I^n$, $R^\infty = \cup_{n \in \mathbb{N}_{>0}} R^n$, $\Sigma^\infty_{\mathsf{com}} = \cup_{n \in \mathbb{N}_{>0}} \Sigma^n_{\mathsf{com}}$ and $\lambda^\infty = \cup_{n \in \mathbb{N}_{>0}} \lambda^n$.

3.2. **Discussion of our modeling choices.** Our definition of RB-systems allows one to model finitely many different process templates because a single process template P can have multiple initial states (representing the disjoint union of the different process templates).

We can easily transform a rendezvous action a involving $j < k$ processes (in particular where $j = 1$, representing an internal transition taken by a single process) into a k-wise rendezvous action by simply adding, for every $j < i \le k$, and every state in P, a self-loop labeled a_i. This transformation works when there are at least k processes in the system.

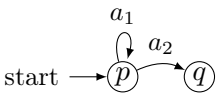


Figure 1: R-template

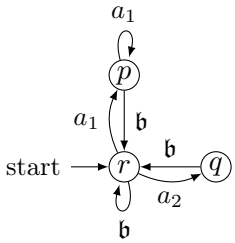


Figure 2: RB-template

This is not a real restriction since all the systems with less than k processes yield a single finite-state system which can be easily model-checked. In any case, all our results hold also if one specifically allows rendezvous actions involving $j < k$ processes.

The assumption that every state in an RB-template is the source of a broadcast edge means that for every configuration f there is a broadcast global-transition with source f.

3.3. **Executions and Specifications.** Take an RB-system $P^n = \langle AP^n, \Sigma^n_{\mathsf{com}}, S^n, I^n, R^n, \lambda^n \rangle$, a path $\pi = t_1 t_2 \dots$ in P^n, and a process i in P^n. Define $proj_\pi(i) := edge_i(t_{j_1}) edge_i(t_{j_2}) \dots$, where $j_1 < j_2 < \dots$ are all the indices j for which $edge_i(t_j) \neq \perp$. Thus, $proj_\pi(i)$ is the path in P taken by process i during the path π. Define the set of *executions* of P^∞, denoted by $\textsc{exec}(P^\infty)$, to be the set of the runs of P^∞ projected onto the state labels of a single process. Note that, due to symmetry, we can assume w.l.o.g. that the runs are projected onto process 1. Formally,

$$\textsc{exec}(P^\infty) = \{\lambda(proj_\pi(1)) \mid \pi \in runs(P^\infty)\},$$

where λ is the labeling of the process template P. We denote by $\textsc{exec-fin}(P^\infty)$ (resp. $\textsc{exec-inf}(P^\infty)$) the finite (resp. infinite) executions in $\textsc{exec}(P^\infty)$.

We present two examples. Note that the letter in a state is both the name of that state as well as the (unique) atom that holds in that state.

Example 3.3. Consider the R-template P in Figure 1. Note that $\textsc{exec-fin}(P^\infty)$ consists of all prefixes of words that match the regular expression pp^*q, and $\textsc{exec-inf}(P^\infty) = \emptyset$.

Example 3.4. Consider the RB-template P in Figure 2. In every run of P^n, every process is involved in at most $n-1$ consecutive rendezvous transitions before a broadcast transition is taken, which resets all processes to the initial state. Thus, $\textsc{exec-fin}(P^\infty)$ consists of all prefixes of words that match $(rr^*(pp^* + q))^*$ since n may be arbitrarily large. Similarly, $\textsc{exec-inf}(P^\infty)$ is the set of words of the form $(rr^*(pp^*+q))^* r^\omega$ and $r^{n_1}(p^{x_1}+q)r^{n_2}(p^{x_2}+q)\dots$, for some sequences $n_1, n_2, \cdots$ and $x_1, x_2, \cdots$ of positive integers such that $\{x_i\}_i$ is bounded. The first form occurs in case process 1 is involved in only finitely many rendezvous transitions, and the second form occurs if process 1 is involved in infinitely many rendezvous transitions. Indeed, in the latter case, the n_is are unconstrained since a broadcast can occur any number of times before a rendezvous, and if $\{x_i\}_i$ is bounded by $B \in \mathbb{N}_{>0}$ then the execution can be realised in the RB-System P^{B+1}.

The definitions above imply the following easy lemma.

Lemma 3.5. *Let P, P' be two RB-templates with the same atomic propositions and edge-labels alphabet.*

(1) *If* $\pi \in runs(P^\infty)$ *and* $\pi' \in runs(P'^\infty)$ *are equi-labeled then* $\lambda(proj_\pi(j)) = \lambda'(proj_{\pi'}(j))$ *for every process* j.
(2) *If* P *and* P' *simulate each other then* $\textsc{exec}(P^\infty) = \textsc{exec}(P'^\infty)$.

Proof. Let $\pi = e_1 e_2 \cdots$ and $\pi' = e'_1 e'_2 \cdots$. For the first item note that for every $i < |\pi|$ the label of the edge e_i is equal to the label of the edge e'_i. It follows that the same processes are active in e_i and e'_i. Thus, the run $proj_\pi(i)$ of the template P is equi-labeled with the run $proj_{\pi'}(i)$ of the template P', and in particular they induce the same sequence of sets of atomic propositions.

For the second item, suppose P' simulates P via $M \subseteq S \times S'$. Derive the relation $C \subseteq S^\infty \times (S')^\infty$ from M point-wise, i.e., $(f, f') \in C$ iff there is $n \in \mathbb{N}_{>0}$ such that i) $f \in S^n, f' \in (S')^n$ and ii) for every $i \leq n$, $(f(i), f'(i)) \in M$. It is routine to check that C is a simulation, and thus P'^∞ simulates P^∞. By a symmetric argument, P^∞ simulates P'^∞. Thus, for every run π in P^∞ there is an equi-labeled run π' in P'^∞, and vice versa. Now apply the first item. □

3.4. **Parameterized Model-Checking Problem.** Specifications represent sets of finite or infinite sequences over the alphabet 2^{AP}. In this work we will consider specifications of finite executions to be given by nondeterministic finite word automata (NFW) and specifications of infinite executions to be given by nondeterministic Büchi word automata (NBW). Standard translations allow us to present specifications in linear temporal logics such as LTL and LTL_f, see Theorem 2.2.

We now define the main decision problem of this work.

Definition 3.6 (PMCP). Let $\mathcal{F}$ be a specification formalism for sets of infinite (resp. finite) words over the alphabet 2^{AP}. The *Parameterized Model Checking Problem* for $\mathcal{F}$, denoted $PMCP(\mathcal{F})$, is to decide, given a process-template P, and a set W of infinite (resp. finite) words specified in $\mathcal{F}$, if all executions in the set $\textsc{exec-inf}(P^\infty)$ (resp. $\textsc{exec-fin}(P^\infty)$) are in W.

Just as for model-checking [Var95], we have three ways to measure the complexity of the PMCP problem. If we measure the complexity in the size of the given template and specification, we have the (usual) complexity, sometimes called *combined complexity*. If we fix the template and measure the complexity with respect to the size of the specification we get the *specification complexity*. If we fix the specification and measure the complexity with respect to the size of the template we get the *program complexity* (we use "program complexity" instead of "template complexity" in order to be consistent with the model-checking terminology). Moreover, if C is a complexity class, we say that the specification complexity of the PMCP-problem is *C-hard* if there is a fixed template such that the induced PMCP problem (that only takes a specification as input) is C-hard in the usual sense; and it is *C-complete* if it is in C and C-hard. Symmetric definitions hold for program complexity.

Results. Our main results solve PMCP for RB-templates for specifications of finite and infinite executions. In both cases we use the automata theoretic approach: given an RB-template P, we show how to build an automaton $\mathcal{M}$ accepting exactly the executions of the RB-system P^∞. Model checking of a specification given by another automaton $\mathcal{M}'$ is thus reduced to checking if the language of $\mathcal{M}$ is contained in the language of $\mathcal{M}'$. The automaton $\mathcal{M}$ will be based on what we call the reachability-unwinding of the given RB-template. In the

finite-execution case $\mathcal{M}$ will be an NFW that is almost identical to the reachability-unwinding. In the infinite execution case $\mathcal{M}$ will be more complicated. Indeed, classic automata over infinite words (e.g., NBW) will not be powerful enough to capture the system (see Lemma 3.8 below) and so we use B-automata; the automaton $\mathcal{M}$ will be based on three copies of the reachability-unwinding (instead of one copy) where from each copy certain edges will be removed based on a classification of edges into different types.

Our first result classifies the complexity for NFW/LTLf specifications:

Theorem 3.7. *Let $\mathcal{F}$ be specifications of sets of finite executions expressed as NFW or LTLf formulas. Then the complexity of $PMCP(\mathcal{F})$ for RB-systems is PSPACE-complete, as is the program complexity and the specification complexity.*

To see that classical acceptance conditions (e.g. Büchi, Parity) are not strong enough for the case of infinite executions, consider the following lemma.

Lemma 3.8. *The process template P in Figure 2 has the property that the set EXEC-INF(P^∞) is not ω-regular.*

Proof. The following pumping argument shows that this language is not ω-regular. Assume by way of contradiction that an NBW $\mathcal{A}$ accepts EXEC-INF(P^∞), and consider an accepting run of $\mathcal{A}$ on the word $(rp^{n+1})^\omega$, where n is the number of states of $\mathcal{A}$. It follows that for each $i \in \mathbb{N}_{>0}$, while reading the i'th block of p's, $\mathcal{A}$ traverses some cycle c_i. Hence, by correctly pumping the cycle c_i, e.g., i times for every $i \in \mathbb{N}_{>0}$, we can obtain an accepting run of $\mathcal{A}$ on a word w' which is not EXEC-INF(P^∞) since it contains blocks of consecutive p's of ever increasing length, contradicting our assumption. □

On the other hand, there is a B-automaton (with a trivial Büchi set) recognizing this language (the counter is incremented whenever p is seen and reset whenever r is seen). This is no accident: we will prove (Theorem 6.6) that for every RB-template P one can build a B-automaton (with a trivial Büchi set) recognizing the infinite executions of P^∞. Combining this with an NBW for the specification, we reduce the model-checking problem to the emptiness problem of a B-automaton. Hence, our second main result provides an EXPTIME upper bound for NBW/LTL specifications:

Theorem 3.9. *Let $\mathcal{F}$ be specifications of sets of infinite executions expressed as NBW or LTL formulas. Then $PMCP(\mathcal{F})$ of RB-systems can be solved in EXPTIME.*

3.5. **Variants with a controller and asymmetric broadcast.** We now give two variants of RB-Systems, i.e., one that incorporate a distinguished "controller" process, and another that allows for asymmetric broadcasts [EFM99].

Given two process templates P_C and P the RB-System with a controller (RBC-System) $P_C \cup P^n$ is the finite LTS $\left\langle AP^{n+1}, \Sigma^{n+1}_{\mathsf{com}}, S^{n+1}, I^{n+1}, R^{n+1}, \lambda^{n+1} \right\rangle$, which is defined exactly as in Definition 3.2, with the only difference that for process 1 we use the process template P_C and for processes 2 to $n+1$ we use process template P. The RBC-System $P_C \cup P^\infty$ is then defined analogously. We now need to adjust the definitions of executions to differentiate between the projection to a controller resp. non-controller processes; we set,

$$\text{EXEC}(P_C \cup P^\infty)_C = \{\lambda(proj_\pi(1)) \mid \pi \in runs(P_C \cup P^\infty)\},$$

and

$$\text{EXEC}(P_C \cup P^\infty) = \{\lambda(proj_\pi(2)) \mid \pi \in runs(P_C \cup P^\infty)\},$$

where, because of symmetry, we can always project to process 2 for a non-controller process.

In order to capture asymmetric broadcasts we need to enhance the edge-labels alphabet of a process template. This is done by introducing a set of broadcast actions Σ_{bcts} (disjoint from Σ_{actn}), and setting the edge-labels alphabet to be $\Sigma_{\mathsf{rdz}} \bigcup \cup_{b \in \Sigma_{\mathsf{bcts}}} \{b_{\mathsf{snd}}, b_{\mathsf{rcv}}\}$. Such a process template is called an *RBA-template* if, in addition, for every state $s \in S$ we have that b_{rcv} is enabled in s for every $b \in \Sigma_{\mathsf{bcts}}$. Given an RBA-template P, the RBA-System P^n is the finite LTS $\langle \mathcal{A}^n, \Sigma^n_{\mathsf{com}}, S^n, I^n, R^n, \lambda^n \rangle$, which is defined as in Definition 3.2 except for the definition of the global transition relation, where we support asymmetric broadcasts instead of symmetric broadcasts as follows:

- *(asymmetric broadcast)* $\sigma = \langle c_1, \ldots, c_n \rangle$ is an n-tuple such that there is some $b \in \Sigma_{\mathsf{bcts}}$ and some i such that $c_i = b_{\mathsf{snd}}$ and $c_j = b_{\mathsf{rcv}}$ for all $j \neq i$, and $f(i) \xrightarrow{c_i} g(i)$ in P, for every $i \in [n]$;

The RBA-System P^∞ is then defined analogously. The set of executions is defined as for RB-Systems.

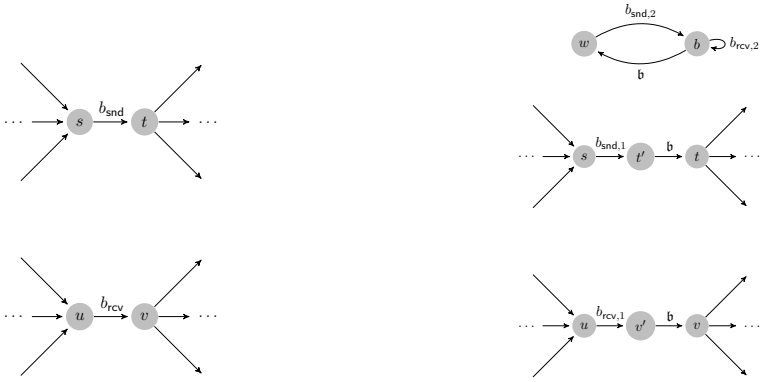
We remark that we define RBA-Systems without a controller for technical convenience. It would be straight-forward to define an RBA-System with a controller in the same way as we did above. However, it is easy to verify that RBA-Systems with a controller are not more powerful than RBA-Systems that lack a controller. That is because having a controller can be simulated through an initial asymmetric broadcast that makes the sender process taking over the role of the controller and the receiver processes continuing as non-controller processes. We now make this statement precise. Recall the notation $(\xi)_a$ and $\xi|_{AP}$ from Section 2.1.

Theorem 3.10. *RBC-Systems and RBA-Systems are equally powerful, more precisely,*

(1) *for each RBC-System, given by process templates P_C and P over atomic propositions AP, we can construct in linear time an RBA-System P' over atomic propositions $AP \cup \{c, p\}$, with $c, p \notin AP$ such that $\textsc{exec}(P_C \cup P^\infty)_C = \{(\xi)_c|_{AP} \mid \xi \in \textsc{exec}(P'^\infty)\}$ and $\textsc{exec}(P_C \cup P^\infty) = \{(\xi)_p|_{AP} \mid \xi \in \textsc{exec}(P'^\infty)\}$;*
(2) *for each RBA-System, given by process template P over atomic propositions AP, we can construct in linear time an RBC System, given by process templates P'_C and P' over atomic propositions $AP \cup \{p\}$ such that $\textsc{exec}(P'^\infty) = \{(\xi)_p|_{AP} \mid \xi \in \textsc{exec}(P'_C \cup P'^\infty)\}$ (the executions of the controller are not important for this statement).*

Proof. For the first item, the symmetric broadcast can be easily implemented by an asymmetric broadcast where the sender simply behaves like the receivers, and a controller can be elected using an initial asymmetric broadcast. Here are the details. The broadcast alphabet Σ_{bcts} consists of two symbols, $\mathfrak{b}$, b: the first for modeling the symmetric broadcast of the RBC-System, and the second to be used for electing the controller. Define the RBA-template $P' = \langle AP \cup \{c, p\}, \Sigma_{\mathsf{rdz}} \cup \{b_{\mathsf{rcv}}, b_{\mathsf{snd}}, \mathfrak{b}_{\mathsf{rcv}}, \mathfrak{b}_{\mathsf{snd}}\}, S', \{\iota\}, R', \lambda' \rangle$ as follows. Let S' consist of the states of P_C, the states of P (assumed to be disjoint from the states of P_C), and a new initial state ι. The transitions relation R' includes all rendezvous transitions of P_C and P, as well as the following new transitions:

- $s \xrightarrow{\mathfrak{b}_{\mathsf{rcv}}} t$ and $s \xrightarrow{\mathfrak{b}_{\mathsf{snd}}} t$ for every transition $s \xrightarrow{\mathfrak{b}} t$ in P_C and P;
- $\iota \xrightarrow{\mathfrak{b}_{\mathsf{rcv}}} \iota$ and $\iota \xrightarrow{\mathfrak{b}_{\mathsf{snd}}} \iota$;
- $\iota \xrightarrow{b_{\mathsf{snd}}} \iota_C$ and $\iota \xrightarrow{b_{\mathsf{rcv}}} \iota_P$ where ι_C and ι_P are the initial states of P_C and P respectively;



(A) RBA templates (sender, top; receivers, bottom)

(B) RBC templates (controller, top; sender, middle; receivers, bottom)

Figure 3: An illustration of how RBA processes can be simulated by RBC processes.

- $s \xrightarrow{b_{\mathsf{rcv}}} s$ for every $s \neq \iota$ (these transitions can never be taken but are added to satisfy the constraint that broadcast can always be received, as required by the definition of RBA-templates).

Finally, let λ_C and λ_P be the labeling functions of P_C and P respectively. Define the labeling function λ' as follows. If s is a state of P then $\lambda'(s) = \lambda_P(s) \cup \{p\}$; if s is a state of P_C then $\lambda'(s) = \lambda_C(s) \cup \{c\}$; and $\lambda'(\iota) = \emptyset$. It is not hard to verify that $\textsc{exec}(P_C \cup P^\infty)_C = \{(\xi)_c|_{AP} \mid \xi \in \textsc{exec}(P'^\infty)\}$ and $\textsc{exec}(P_C \cup P^\infty) = \{(\xi)_p|_{AP} \mid \xi \in \textsc{exec}(P'^\infty)\}$.

For the second item in the statement of the theorem, the idea is depicted in Figure 3. Intuitively, to simulate an asymmetric broadcast, we use a symmetric broadcast preceded by a rendezvous with the controller, which ensures that there is single sender and multiple receivers. For an RBA template $P = \langle AP, \Sigma_{\mathsf{rdz}} \bigcup \cup_{b \in \Sigma_{\mathsf{bcts}}} \{b_{\mathsf{snd}}, b_{\mathsf{rcv}}\}, S, I, R, \lambda \rangle$, we construct two process templates

$$P'_C = \left\langle AP \cup \{p\}, \Sigma'_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S'_C, I'_C, R'_C, \lambda'_C \right\rangle$$

and

$$P' = \left\langle AP \cup \{p\}, \Sigma'_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S', I', R', \lambda' \right\rangle,$$

as follows. The new rendezvous actions consist of the old rendezvous actions as well as $b_{\mathsf{snd}}, b_{\mathsf{rcv}}$ for every broadcast action $b \in \Sigma_{\mathsf{bcts}}$; thus, we have

$$\Sigma'_{\mathsf{rdz}} = \Sigma_{\mathsf{rdz}} \bigcup \cup_{b \in \Sigma_{\mathsf{bcts}}} \{b_{\mathsf{snd},1}, b_{\mathsf{snd},2}, b_{\mathsf{rcv},1}, b_{\mathsf{rcv},2}\}$$

(recall the remarks in Section 3.2 that allow us to have rendezvous actions involving any number of processes).

We now define the components of the controller P'_C:

- $S'_C = \{w, \bot_C\} \cup \Sigma_{\mathsf{bcts}}$, where w (the "waiting" state) and $\bot_C$ (the "dead" state) are new states;
- $I'_C = \{w\}$;
- $\lambda'_C(s) = \emptyset$ for all $s \in S'_C$ (the labeling function of P'_C is not important);
- the transitions in R'_C consist of rendezvous transitions $w \xrightarrow{b_{\mathsf{snd},2}} b$ and $b \xrightarrow{b_{\mathsf{rcv},2}} b$ for every $b \in \Sigma_{\mathsf{bcts}}$, and the symmetric broadcast transitions $b \xrightarrow{\mathfrak{b}} w$ and $w \xrightarrow{\mathfrak{b}} \bot_C$ and $\bot_C \xrightarrow{\mathfrak{b}} \bot_C$.

We now define the components of the process template P':

- let S' consist of the states in S, a new state $\perp$, and new ("intermediate") states of the form q' where q varies over the states of P that are the destination of some asymmetric broadcast transition in P;
- $I' = I$;
- $\lambda'(s) = \lambda(s) \cup \{p\}$ for $s \in S$, and $\lambda'(q') = \emptyset$ for $q' \in S' \setminus S$ (in particular, $\lambda'(\perp) = \emptyset$);
- the transitions R' include all the rendezvous transitions of P, the rendezvous transition $s \xrightarrow{b_{\mathsf{snd},1}} t'$ for every asymmetric send-broadcast transition $s \xrightarrow{b_{\mathsf{snd}}} t$ of P, and the rendezvous transition $u \xrightarrow{b_{\mathsf{rcv},1}} v'$ for every asymmetric receive-broadcast transition $u \xrightarrow{b_{\mathsf{rcv}}} v$ of P. Further, R' includes, for every $q' \in S'$, the symmetric broadcast transition $q' \xrightarrow{\mathfrak{b}} q$, and for every state $s \in S$, the symmetric broadcast transitions $s \xrightarrow{\mathfrak{b}} \perp$, and the symmetric broadcast transition $\perp \xrightarrow{\mathfrak{b}} \perp$.

The states $\perp_C, \perp$ are not drawn in Figure 3.

We now argue that $\text{EXEC}(P^\infty) = \{(\xi)_p|_{AP} \mid \xi \in \text{EXEC}(P'_C \cup P'^\infty)\}$. In what follows, although the n processes over the template P' in the RBA-system $P'_C \cup P'^n$ are numbered $2, \cdots, n+1$, we will number them $1, \cdots, n$, and refer to the process over template P'_C as "the controller" instead of as process 1.

We first argue the inclusion $\text{EXEC}(P^\infty) \subseteq \{(\xi)_p|_{AP} \mid \xi \in \text{EXEC}(P'_C \cup P'^\infty)\}$: Given a run π of P^n (for some n), we will construct a corresponding run π' of $P'_C \cup P'^n$. Every rendezvous transition in π is simulated in π' by the corresponding rendezvous transition of P'^n. Every asymmetric broadcast transition with action $b \in \Sigma_{\mathsf{bcts}}$ in π is simulated in π' by a sequence of transitions of $P'_C \cup P'^\infty$, as follows. Suppose process j in the RBA-system is sending the asymmetric broadcast by taking the edge $s \xrightarrow{b_{\mathsf{snd}}} t$ in P. Then, process j in the RBC-system rendezvouses with the controller on the action b_{snd}, i.e., process j takes the edge $s \xrightarrow{b_{\mathsf{snd},1}} t'$ in P', and the controller takes the edge $w \xrightarrow{b_{\mathsf{snd},2}} b$ in P'_C. Further, consider the remaining processes in the RBA-system that are receiving the broadcast (in some arbitrary order), and suppose process i it takes the edge $u \xrightarrow{b_{\mathsf{rcv}}} v$ in P. Then, that process in the RBC-system rendezvouses with the controller on action b_{rcv}, i.e., process i takes the edge $u \xrightarrow{b_{\mathsf{rcv},1}} v'$ in P', and the controller takes the edge $b \xrightarrow{b_{\mathsf{rcv},2}} b$ in P'_C. Once all the receiving processes in the RBA-system have been accounted for in this way, the RBC-system does a symmetric broadcast, so the controller returns to the waiting state w (i.e., it takes the edge $b \xrightarrow{\mathfrak{b}} w$ in P'_C), and all the non-controller processes go to the target of the broadcast transition (i.e., a process in state v' takes the edge $v' \xrightarrow{\mathfrak{b}} v$ in P'). In this way, the runs π, π' agree on the projection onto a single (non-controller) process when the intermediate states, whose label does not include the atom p, are removed.

We now argue $\text{EXEC}(P^\infty) \supseteq \{(\xi)_p|_{AP} \mid \xi \in \text{EXEC}(P'_C \cup P'^\infty)\}$: given a run π' of $P'_C \cup P'^n$ (for some n), we show there is a corresponding run π of P^n. Unlike the previous inclusion, this time our construction will be tailored to faithfully simulate the behavior of only a single process, i.e., π will depend on the process of interest. Recall from Section 3.3 that, due to symmetry, we can restrict our attention to process 1 (which, by our note above, is a process with template P resp. P', and not the controller).

We claim that, w.l.o.g., π' is structured as a sequence of blocks of the form: first, zero or more rendezvous transitions that do not involve the controller; then, unless there is no

further symmetric broadcast, k many rendezvous transitions that do involve the controller for some k with $1 \leq k \leq n$ (the first of these is on an action b_{snd}, and the remainder on b_{rcv}), at least one of which involves process 1, followed by a symmetric broadcast. To see why this claim holds, first note that we can swap the order of successive transitions if the first of these is a rendezvous between the controller and a process, say process j, and the second is a rendezvous without the controller, say amongst processes X (the reason for this is that the rendezvous with the controller puts process j into an intermediate state, from which there is no rendezvous edge, and thus j is not in X, and thus the global state, as well as the projection onto process 1, after these transitions is not changed by the swap). Second, consider the transitions between two symmetric broadcasts. Repeatedly swap as above until all rendezvous transitions that do not involve the controller (if any) are before all rendezvous transitions that do (if any). This completes the block, unless process 1 does not occur in a rendezvous with the controller. But in this case we can safely trim π' after the last transition in which process 1 is active since the subsequent symmetric broadcast puts process 1 in the sink $\perp$ that has an empty label. Third, if there are finitely many symmetric broadcasts, consider the (possibly infinitely many) transitions after the last symmetric broadcast (if there are no symmetric broadcasts, then consider all the transitions). Note that there are at most n such transitions that rendezvous with the controller. Since a process that rendezvous with the controller is put into a state where it cannot rendezvous with any other processes, we can repeatedly remove the last such transition until there are none left (note that even if process 1 is involved in such a transition, it is put into an intermediate state by such a transition, which has empty label, and thus can safely be removed).

It should be now quite clear how to simulate each block of π': every rendezvous transition that does not involve the controller is simulated by a corresponding rendezvous transition in P^n; the rest of the block, if any, can be simulated by a single asymmetric broadcast transition of P^n. Note, however, that processes in P'^n that do not rendezvous with the controller in this block will no longer be faithfully simulated since such processes will transition to the dead state $\perp$ on the symmetric broadcast. This is not a problem since these processes will never again participate in any rendezvous transitions, and thus will have no influence on the ability of the projection of π on process 1 to faithfully capture the projection of π' on process 1. Hence, we are free to define the target states in π of these processes during future asymmetric broadcasts, if any (note that by assumption at least one target state exists for every asymmetric broadcast). □

The PMCP for RBA-Systems is quite well understood [EFM99], i.e., it is undecidable for ω-regular specifications, and decidable for regular specifications. We thus get:

Theorem 3.11.

(1) *Let $\mathcal{F}$ be specifications of sets of infinite executions expressed as NBW or LTL formulas. Then $PMCP(\mathcal{F})$ of RBC-systems is undecidable.*
(2) *Let $\mathcal{F}$ be specifications of sets of finite executions expressed as NFW or LTLf formulas. Then $PMCP(\mathcal{F})$ of RBC-systems is decidable.*

Proof. By Theorem 3.10, one can transfer verification tasks between RBA- and RBC-Systems: This can be achieved by a formula/automaton transformation, and the verification of the transformed formula/automaton on the transformed system. We exemplify how to transfer LTL/LTLf specifications: Given an LTL/LTLf specification ϕ, we implement the projection operation $(\cdot)_p$ by replacing every occurrence of an atomic proposition X in ϕ by $\neg p \,\mathsf{U}(p \wedge X)$,

resulting in a formula ϕ'; in order to only consider executions that do not reach the dead state we can then consider the specification $(\mathsf{G}\,\neg p\,\mathsf{U}\,p) \rightarrow \phi'$. Similar ideas can be implemented through automata transformations. In particular, the undecidability (resp. decidability) of ω-regular (resp. regular) specifications for RBA-systems transfers to RBC-systems. □

3.6. **Large systems simulating small systems.** We provide a simple but useful property of RB-systems that will be used throughout the rest of this paper. Intuitively, a large RB-system can, using a single run, partition its processes into several groups, each one simulating a run of a smaller RB-system, *as long as all the simulated runs have the same number of broadcasts*. In order to state and prove this result, we need the following.

Notation. Let $X \subseteq [n]$ be a set of processes. For a configuration $f : [n] \rightarrow S$ of P^n define $f\big|_X$ to be the restriction of f to the domain X. Similarly, for a global transition t of P^n, say $f \xrightarrow{\sigma} g$, if t is a broadcast transition (i.e., $\sigma = \mathfrak{b}$), or a rendezvous transition whose active processes are all in X (i.e., $\sigma \neq \mathfrak{b}$ and $\mathsf{active}(t) \subseteq X$), then we define $t\big|_X$ to be $f\big|_X \xrightarrow{\sigma} g\big|_X$; otherwise (i.e., if $\sigma \neq \mathfrak{b}$ and $\mathsf{active}(t) \not\subseteq X$), then $t\big|_X$ is undefined. Finally, given a path π in P^n, if for every $1 \leq i \leq |\pi|$ we have that $\mathsf{active}(\pi_i) \subseteq X$ or $\mathsf{active}(\pi_i) \subseteq [n] \setminus X$, then the restriction $\pi\big|_X := \pi_{i_1}\big|_X \pi_{i_2}\big|_X \dots$ is defined by taking $i_1 < i_2 < \dots$ to be exactly the indices $1 \leq j \leq |\pi|$ for which $\pi_j\big|_X$ is defined; otherwise (i.e., if there is a transition on π in which some of the active processes are in X and some are not in X) $\pi\big|_X$ is undefined.

We will implicitly rename processes as follows. Let $rename : X \rightarrow [|X|]$ be a bijection. Consider configurations f, transitions t, and paths π of P^n. By renaming the processes using $rename$ we can think of $f\big|_X$ as a configuration of $P^{|X|}$, and $t\big|_X$ (if defined) as the transition of $P^{|X|}$ obtained by restricting the configurations f and g in t to X, and $\pi\big|_X$ (if defined) as a path of $P^{|X|}$.

For a process template P, paths $\pi_1, \pi_2, \cdots, \pi_h$ in P^∞ (possibly using different numbers of processes), and pairwise disjoint subsets $X_1, X_2, \cdots, X_h$ of $\mathbb{N}_{>0}$, we say that a path π in P^∞ *simulates* $\pi_1, \cdots, \pi_h$ (with $X_1, X_2, \cdots, X_h$) if $\pi\big|_{X_i} = \pi_i$ for every i. Observe that, if $\pi_1, \cdots, \pi_h$ do not have the same number of broadcasts then there is no π that can simulate them. The next lemma shows that this condition is not only necessary but also sufficient.

Lemma 3.12 (Composition)**.** *Given an integer b, paths (resp. runs) $\pi_1, \dots, \pi_h$ in RB-systems $P^{n_1}, \dots, P^{n_h}$ each with exactly b broadcast transitions: for every $n \geq \Sigma_{i=1}^h n_i = m$, every configuration f in P^n and pairwise disjoint subsets $X_1, X_2, \cdots, X_h$ of $\mathbb{N}_{>0}$ such that $f\big|_{X_i} = \mathsf{src}(\pi_i)$ for every i, there exists a path (resp. run) π in P^n starting in f that simulates $\pi_1, \cdots, \pi_h$ with $X_1, X_2, \cdots, X_h$.*

Proof. We begin by proving the lemma in the special case of R-systems. For every $j \in [h]$, we will have the n_j processes in the set X_j simulate $\pi_j = e_{j,1}\ e_{j,2} \dots$. The extra processes (between $m+1$ and n) do not move. Note that all transitions on $\pi_1, \dots, \pi_h$ are rendezvous involving k processes. Whenever a rendezvous appearing on π_j is performed in P^n only k processes in X_j move, leaving the others unaffected. Thus, π can be obtained by any interleaving of the rendezvous appearing on $\pi_1, \dots, \pi_h$ as long as the relative internal ordering of rendezvous on each of these paths is maintained (e.g., round-robin $e_{1,1}\ e_{2,1} \dots e_{h,1}\ e_{1,2}\ e_{2,2} \dots e_{h,2} \dots$).

Now, we consider the case of general RB-systems. As before, for every $j \in [h]$, we will have the n_j processes in the set X_j simulate π_j. If $n > m$, the extra processes are ignored (however, they do move whenever there is a broadcast). Each path $\pi_1, \ldots, \pi_h$ is cut into $b+1$ segments (numbered $0, \ldots, b$), each containing only rendezvous transitions and followed by a broadcast transition. Thus, the i'th segment of each path is followed by the $(i+1)$'th broadcast. The path π is constructed in $b+1$ phases: in phase i, the i'th segment of all the paths $\pi_1, \ldots, \pi_h$ are simulated as was done in the R-systems case, followed (if $i < b$) by a single broadcast transition that forces the simulation of the i'th broadcast on all of these paths at once. □

We now present a more flexible form of simulation in which the processes that are assigned to simulate a given path are not fixed throughout the simulation (this will be used in the proof of Theorem 6.6).

Definition 3.13. We say that π_0 *weakly-simulates* $\pi_1, \cdots, \pi_h$ if there exists an integer l and a decomposition of each of these paths into l segments, the i'th segment of π_j is denoted π_j^i for $1 \le i \le l, 0 \le j \le h$, and pairwise disjoint sets $X_1^i, \cdots, X_h^i$ for $1 \le i \le l$, such that for every i we have that π_0^i simulates $\pi_1^i, \cdots, \pi_h^i$ (with $X_1^i, X_2^i, \cdots, X_h^i$).

The difference between weak-simulation and simulation is that the set of processes simulating each path may be changed at the end of each segment. The following observation follows immediately from the definition above.

Remark 3.14. If π_0 weakly-simulates *cycles* $\pi_1, \cdots, \pi_h$, then $\mathsf{dst}(\pi_0)\big|_{X_j^l} = \mathsf{dst}(\pi_j) = \mathsf{src}(\pi_j)$ for every j. In words: for every simulated cycle π_j, the destination configuration of the weakly simulating path π_0 restricted to the set of processes X_j^l (the states used in simulating the last segment of π_j) is equal to the destination configuration of π_j and thus, since π_j is a cycle, also to its source configuration.

4. Discrete Timed Networks

In this section we give the formal definition of a *discrete timed network*, with minor changes compared with [ADM04]. In particular, we first describe the form of a process template and later the operational semantics defining how networks of such processes evolve. In this work, unless stated otherwise, we only consider timed networks *without* a controller, and always assume a discrete time model $\mathbb{N}$.

Definition 4.1. A *timed-network (TN) template* is a tuple $\langle A, C, grd, rst, CP\rangle$ where $A = \langle AP, \Sigma_{\mathsf{rdz}}, S, I, R, \lambda\rangle$ is a finite LTS, C is a finite set of *clock variables* (also called *clocks*), each transition $t \in R$ is associated with a *guard* $grd(t)$ and a *reset command* $rst(t)$, and CP is a set of *clock predicates*, i.e., predicates of the form $x \bowtie c$ where $x \in C$, $c \in \mathbb{N}$ is a constant, and $\bowtie \in \{>, =\}$. A guard is a Boolean combination of clock predicates. A reset command is a subset of C.

The *size* of a TN template is the size of the LTS A (i.e., the number of states plus the number of transitions) plus the sizes of all the guards, reset commands, and clock predicates where the constants in the clock predicates are represented in *unary*.[4]

[4]The unary representation is chosen in order to elicit the relation to RB-systems, i.e., this representation allows us to show that the PMCP for timed-networks and RB-systems is polynomial-time inter-reducible.

A timed network T^n consists of the parallel composition of $n \in \mathbb{N}_{>0}$ template processes, each running a copy of the template. Each copy has a *local configuration* (q, K), where $q \in S$ and $K : C \to \mathbb{N}$ is a clock evaluation mapping each clock to its (discrete) value. We say that an evaluation $K : C \to \mathbb{N}$ *satisfies* a Boolean combination of clock predicates ϕ, if ϕ evaluates to true when every occurrence of clock x in ϕ is replaced by the value $K(x)$. A rendezvous action a is *enabled* if there are k processes $i_1, \cdots, i_k$ such that for every $j \in [k]$ process i_j is in a local configuration (q_j, K_j) for which there is an edge $q_j \xrightarrow{a_j} q'_j$, say t_j, and the clock evaluation K_j satisfies the guard $grd(t_j)$. The rendezvous action is *taken* means that the k processes change their local configurations to (q'_i, K'_i), where K'_i is obtained from K_i after setting the values of the clocks in $rst(t_i)$ to 0. Besides these rendezvous transitions, the system can evolve by taking timed-transitions in which all clocks of all processes advance by one time unit (so every $K(x)$ increases by one).[5] Runs of T^n projected onto a single process induce sequences over the alphabet $2^{AP \cup CP}$ of the atomic predicates and clock predicates that hold at each local configuration. Specifications (for the behavior of a single process) can be given as automata or linear-temporal properties over the alphabet $2^{AP \cup CP}$.

To formally define a timed network as an LTS, its executions, and its corresponding PMCP, one can proceed by instantiating the intuitive description given above, along the lines of, e.g., [ADM04]. Alternatively, one can give an equivalent definition (in the sense that it yields exactly the same LTS for the timed network, and thus also the same set of executions and PMCP) by observing that timed networks are essentially RB-systems whose RB-template P_T is induced by the given TN-template T by viewing local configurations as states of P_T, and thinking of timed transitions as symmetric broadcast transitions. Notice that following this approach, the obtained RB-template P_T would be infinite, due to clocks potentially increasing without bounds. In order to make it finite, one can simply truncate clock values up to an appropriate upper bound. In the following we give a detailed construction.

Defining Timed systems as RB-systems. [6] Let T be a TN-template $\langle A, C, grd, rst, CP\rangle$ where $A = \langle AP, \Sigma_{\mathsf{rdz}}, S, I, R, \lambda\rangle$. Define the infinite RB-template

$$P_T = \langle AP \cup CP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S_T, I_T, R_T, \lambda_T\rangle$$

where

- $S_T = S \times \mathbb{N}^C$ is the set of *local configurations*,
- I_T consists of all pairs (q, K) where $q \in I$ and $K(x) = 0$ for all $x \in C$,
- R_T consists of two types of transitions:
  - *timed transitions* of the form $(q, K) \xrightarrow{\mathfrak{b}} (q, K')$, for every $q \in S$ and $K : C \to \mathbb{N}$, and such that $K'(x) = K(x) + 1$ for all $x \in C$; or
  - *rendezvous transitions* of the form $(q, K) \xrightarrow{\sigma} (q', K')$, for every transition $t = (q, \sigma, q') \in R$, for every evaluation $K : C \to \mathbb{N}$ satisfying the guard $grd(t)$, and such that for every $x \in C$, if $x \in rst(t)$ then $K'(x) = 0$ and otherwise $K'(x) = K(x)$.
- $\lambda_T \subseteq S_T \times (AP \cup CP)$ consists of all pairs $((q, K), p)$ such that either $p \in AP$ and $p \in \lambda(q)$, or $p \in CP$ and K satisfies the clock predicate p.

[5] Alternatively, as in [ADM04], one can let time advance by any amount.

[6] While Definition 3.1 requires an RB-template to be finite, for the purpose of this section we lift this restriction.

Given a TN-template $T = \langle A, C, grd, rst, CP\rangle$ with $A = \langle AP, \Sigma_{\mathsf{rdz}}, S, I, R, \lambda\rangle$, and $n \in \mathbb{N}_{>0}$, we define the *timed network* T^n, composed of n processes, to be the RB-system $(P_T)^n$, and the *timed network* T^∞ to be the RB-system $(P_T)^\infty$.

Definition 4.2 (PMCP for Timed-Networks). Let $\mathcal{F}$ be a specification formalism for sets of infinite (resp. finite) words over the alphabet $2^{AP\cup CP}$. The *Parameterized Model Checking Problem for Timed-Networks* for $\mathcal{F}$, denoted $PMCP(\mathcal{F})$, is to decide, given a TN-template T, and a set L of infinite (resp. finite) words specified in $\mathcal{F}$, if all executions in the set EXEC-INF(T^∞) (resp. EXEC-FIN(T^∞)) are in L.

In Sections 5 and 6 we show how to solve PMCP for finite RB-templates for specifications of finite and infinite executions respectively. This cannot be used directly to solve the PMCP for timed networks since given a timed template T the RB-template P_T is infinite. However, the next Lemma shows that given T, there is a finite RB-template U such that EXEC(T^∞) = EXEC(U^∞). The template U is obtained from P_T by clipping the clock values to be no larger than 1 plus the maximal constant appearing in the clock predicates CP.

Lemma 4.3. *Let T be a TN-template and let $d = \max\{c : x \bowtie c \in CP\}+1$. One can construct in time polynomial the size of T a finite RB-template U such that* EXEC(T^∞) = EXEC(U^∞).

Proof. We use the following clipping operation: for $d \in \mathbb{N}_{>0}$ and $K : C \to \mathbb{N}$ let $clip_d(K) : C \to \{0, \ldots, d\}$ map x to $\min\{K(x), d\}$. For a local configuration (q, K) define $clip_d(q, K)$ to be $(q, clip_d(K))$, and extend this to sets of configurations point-wise. Let t be any transition $(q, K) \xrightarrow{\sigma} (q', K')$, define $clip_d(t)$ to be $clip_d(q, K) \xrightarrow{\sigma} clip_d(q', K')$, and extend this to sets of transitions point-wise.

Note that, by our choice of d, an evaluation K satisfies a Boolean combination of clock predicates ϕ iff the evaluation $clip_d(K)$ satisfies ϕ.

Let T be a TN-template $\langle A, C, grd, rst, CP\rangle$ where $A = \langle AP, \Sigma_{\mathsf{rdz}}, S, I, R, \lambda\rangle$, and $P_T = \langle AP \cup CP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S_T, I_T, R_T, \lambda_T\rangle$. Then let

$$U = \left\langle AP \cup CP, \Sigma_{\mathsf{rdz}}, S', I', R', \lambda' \right\rangle$$

where

- $S' = clip_d(S_T)$,
- $I' = clip_d(I_T)$,
- $R' = clip_d(R_T)$, and
- $\lambda' = \{((q, clip_d(K)), p) : ((q, K), p) \in \lambda_T\}$.

We claim that EXEC(T^∞) = EXEC(U^∞). To see this note that P_T and U are bisimilar using the relation B defined by letting $((q, K), (q', K')) \in B$ iff $(q', K') = clip_d(q, K)$. It is not hard to see by following the definitions that B is a bisimulation relation. To finish apply Lemma 3.5 item 2. □

The construction used in Lemma 4.3 is illustrated in Figure 4. We note that the polynomial-time result crucially depends on constants represented in unary (note that the construction polynomially depends on $d = \max\{c : x \bowtie c \in CP\} + 1$). We leave the investigation of complexity-theoretic consideration when numbers are represented in binary for future work.

We next show that RB-systems are not more powerful than timed networks. We show that, by allowing for operations that take a subsequence and remove atomic propositions, RB-systems and timed networks can define the same languages of (finite and infinite) executions. Recall the notation $(\xi)_a$ and $\xi|_{AP}$ from Section 2.1.

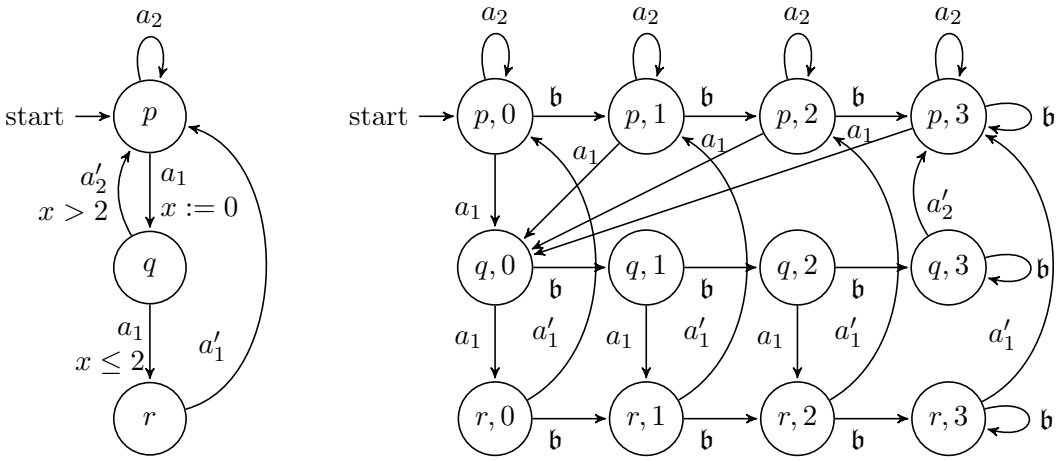


Figure 4: Construction from Lemma 4.3. A TN-template T (left) with one clock x, and the RB-template U (right) with $d = 3$. For readability, atomic predicates and clock predicates are not drawn.

Lemma 4.4. *Let P be a process template with atomic propositions AP. Then, one can construct in linear time a TN-template T with a singleton set of clocks $C = \{c\}$, clock predicates $CP = \{c = 0, c = 1\}$, and atomic proposition $AP \cup CP$, such that $\text{EXEC}(P^\infty) = \{(\pi)_{c=0}|_{AP} \mid \pi \in \text{EXEC}(T^\infty)\}$.*

Proof. We consider the process template $P = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S, I, R, \lambda\rangle$ and construct the TN-template $T = \langle A, C, \mathit{grd}, \mathit{rst}, CP\rangle$, where $C = \{c\}$ and $CP = \{c = 0, c = 1\}$, and the LTS $A = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}, \sharp\}, S, I, R_A, \lambda\rangle$ is obtained from P as follows:

(1) R_A contains every rendezvous transition t of P, where we set $\mathit{grd}(t) := c = 0$ for the guard and $\mathit{rst}(t) := \{\}$ for the reset.

(2) R_A contains an internal transition $s \xrightarrow{\sharp} s'$ for every broadcast transition $s \xrightarrow{\mathfrak{b}} s'$ of P, where we set $\mathit{grd}(t) := c = 1$ for the guard, and $\mathit{rst}(t) := \{c\}$ for the reset (internal transitions are 1-rendezvous transitions where a single process changes state).

We claim that $\text{EXEC}(P^\infty) = \{(\xi)_{c=0}|_{AP} \mid \xi \in \text{EXEC}(T^\infty)\}$. To show that, we will need the following notation: given a configuration f of P^n, for some n, and a clock value $v \in \{0, 1\}$, we denote by f^v the configuration of T^n obtained from f by defining the clock value of each of the processes to be v (i.e., by letting $f^v(j) = (f(j), (v))$, for every $j \in [n]$). Also, it will be convenient to call a transition $f \xrightarrow{(j,\sharp)} g$ of T^n an internal transition (of process j).

We first show that $\text{EXEC}(P^\infty) \subseteq \{(\xi)_{c=0}|_{AP} \mid \xi \in \text{EXEC}(T^\infty)\}$. Given $\pi' \in \mathit{runs}(P^\infty)$, let n be such that $\pi' \in \mathit{runs}(P^n)$. We will construct a corresponding run $\pi \in \mathit{runs}(T^n)$. Intuitively, a rendezvous transition of π' is simulated directly by a single corresponding transition in π whose source and destination have all clocks at zero; whereas a broadcast transition of π' is simulated in π by a sequence of transitions: one timed transition (that increases all clocks from zero to one), followed by one internal transition of each process. Observe that, by incrementing the clocks to one, the timed transition enables the guards of the internal transitions of each of the processes, and that each such internal transition, once taken, resets the clock back to zero.

Formally, we construct π by considering the transitions of π' in order. Let $i' \geq 1$, assume that we already constructed a prefix of length $i-1$ of π that simulates the first $i'-1$ transitions of π' (initially, this prefix is obviously empty). The construction will maintain the invariant $\mathsf{dst}(\pi'_{i'})^0 = \mathsf{dst}(\pi_i)$; here we use the notation f^v introduced above. Consider the two cases for the transition $\pi'_{i'}$. (i) if $\pi'_{i'} = (f', \sigma, g')$ is a rendezvous transition, then extend π by letting $\pi_i = (f'^0, \sigma, g'^0)$. (ii) if $\pi'_{i'} = (f', \mathfrak{b}, g')$ is a broadcast transition, then extend π as follows: first add a timed transition $\pi_i = (f'^0, \mathfrak{b}, f'^1)$; then add a series of n transitions, one for each process $j \in [n]$, in which process j takes the internal transition from $f'(j)$ to $g'(j)$ (i.e., let $f_i = f'^1$, and let $\pi_{i+j} = (f_{i+j-1}, ((j, \sharp)), f_{i+j})$, where f_{i+j} is identical to f_{i+j-1} except that $f_{i+j}(j) = (g'(j), 0)$); Observe that at the end of this sequence we reach the configuration g'^0. It is easy to verify that $\lambda(proj_{\pi'}(i)) = (\lambda(proj_{\pi}(i)))_{c=0}|_{AP}$ for all processes i. Hence, $\textsc{exec}(P^\infty) \subseteq \{(\pi)_{c=0}|_{AP} \mid \pi \in \textsc{exec}(T^\infty)\}$.

We now show the reverse inclusion $\{(\xi)_{c=0}|_{AP} \mid \xi \in \textsc{exec}(T^\infty)\} \subseteq \textsc{exec}(P^\infty)$. Given $\pi \in runs(T^\infty)$, let n be such that $\pi \in runs(T^n)$. We will construct a corresponding run $\pi' \in runs(P^n)$. Unlike the previous inclusion, this time our construction will be tailored to faithfully simulate the behavior of only a single process, i.e., π' will depend on the process of interest. Recall from Section 3.3 that, due to symmetry, we can restrict our attention to process 1, i.e., it is enough to show that †: $\lambda(proj_{\pi'}(1)) = (\lambda(proj_{\pi}(1)))_{c=0}|_{AP}$.

We first claim that, w.l.o.g., π is structured as a sequence of blocks of the form: one or more multi-process rendezvous transitions, followed (unless π has no more timed transitions) by a timed transition and $1 \leq k \leq n$ internal transitions of k different processes, one of which is process 1. To see why this claim holds, let $i_1 < i_2 < \cdots$ be the positions of timed transitions of π. Observe that, for every h, all the clocks in $\mathsf{dst}(\pi_{i_h})$ are at least 1 and thus, due to the way the transitions of T are guarded, for every $j \in [n]$, the first transition in which process j can be active after π_{i_h} and before $\pi_{i_{h+1}}$ must be an internal transition of j (in which, obviously, no other process is active). Hence, we can push all the internal transitions to immediately follow the timed transition π_{i_h} without changing the projection of π on any single process. Finally, observe that if process 1 did not make an internal transition between π_{i_h} and $\pi_{i_{h+1}}$ then we can safely trim π and ignore everything after position i_h. Indeed, in this case at $\mathsf{dst}(\pi_{i_{h+1}})$ process 1 has a clock value > 1, and thus (since all non-timed transitions are guarded by $c = 0$ or $c = 1$), this value will never go below 1 again. Now, recall that † restricts our attention to the projection of π on process 1 transitions whose sources have clock value 0.

It should be now quite clear how to simulate each block of π: every multi-process rendezvous transition can be easily simulated by a corresponding rendezvous transition in P^n, whereas the rest of the block (i.e., the timed transition followed by a sequence of internal transitions) can be simulated by a single broadcast transition of P^n. Note, however, that processes which do not make an internal transition in the block will not be faithfully simulated since such processes never change their state in this block (only their clock changes during by the timed transition), whereas the simulating broadcast may unfaithfully change their state. However, as we noted above, these processes will not include process 1, nor can they rendezvous with any process (again, because their guards will always be false), and thus will have no influence on the ability of the projection of π' on process 1 to faithfully capture the projection of π on process 1.

Given a configuration f of T^n, denote by $state(f) \in S^n$, the configuration in P^n obtained from it by dropping the clock values of all processes. Formally, we construct π' by considering the transitions of π in order, block by block. Let $i \geq 1$, and assume that we already

constructed a prefix of length $i'-1$ of π' that simulates the first $i-1$ transitions of π. Whenever π_i is a multi-process rendezvous transition, or is the last transition in a block, the construction will maintain the invariant that the state of process j in $\mathsf{dst}(\pi_i)$ is equal to its state in $\mathsf{dst}(\pi'_{i'})$, for every process j whose clock value at $\mathsf{dst}(\pi_i)$ is 0. Consider the following two cases: (i) $\pi_i = (f, \sigma, g)$ is a multi-process rendezvous transition. Then, extend π' by letting $\pi'_{i'} = (state(f), \sigma, state(g))$. (ii) π_i is a timed transition (let $f' = state(\mathsf{dst}(\pi_i))$), and the rest of the transitions in the block are $k \leq n$ internal transitions of different processes, i.e., there is a set of processes $X = \{x_1, x_2, \cdots, x_k\} \subseteq [n]$, and for every $j \in [k]$ there is an internal transition $f'(j) \xrightarrow{\sharp} q_j$ of the template T, such that $\pi_{i+j} = f_{i+j-1} \xrightarrow{(x_j,\sharp)} f_{i+j}$, where $f_i = \mathsf{dst}(\pi_i)$, and f_{i+j} is identical to f_{i+j-1} except that $f_{i+j}(x_j) = (q_j, 0)$. Then, extend π' by letting $\pi'_{i'} = (f', \mathfrak{b}, g')$ be a transition of P^n satisfying $g'(x_j) = q_j$ for every $j \in [k]$ (note that the exact broadcast transitions taken by processes outside X are unimportant). It is not hard to verify that † holds, as promised. □

We note that Lemma 4.4 implies that for every specification φ (given as LTL formula resp. finite automaton) one can construct a specification φ' such that all executions EXEC-INF(P^∞) (resp. EXEC-FIN(P^∞)) satisfy φ iff all executions EXEC-INF(T^∞) (resp. EXEC-FIN(T^∞)) satisfy the specification φ'; this is because the operation $(\pi)_a$ can be implemented as a formula resp. automaton transformation.

From Lemma 4.3 and Lemma 4.4 we immediately obtain the main result of this section:

Theorem 4.5. *The Parameterized Model Checking Problems for RB-Systems and Timed-Networks are polynomial-time inter-reducible; in particular, (lower as well as upper) bounds on the program/specification/combined complexity transfer.*

Timed Networks with a Controller. The inter-reducibility between RB-Systems and Timed Networks extends to systems with a controller as we sketch in the following. Given two TN-templates T_C and T, the TN-System with a controller (TNC-System) $T_C \cup T^n$ is then defined as the RBC-System $P_{T_C} \cup P_T^n$, where P_{T_C} and P_T are the induced (infinite) RB-templates introduced above. The TNC-System $T_C \cup T^\infty$ is then defined analogously. Lemma 4.3 and Lemma 4.4 can then straightforwardly be extended to the relationship between TNC-Systems and RBC-Systems. This gives us the following results:

Theorem 4.6. *The Parameterized Model Checking Problems for RBC- and TNC-Systems are polynomial-time inter-reducible; in particular, (lower as well as upper) bounds on the program/specification/combined complexity transfer.*

Corollary 4.7.

(1) *Let $\mathcal{F}$ be specifications of sets of infinite executions expressed as NBW or LTL formulas. Then $PMCP(\mathcal{F})$ of TNC-Systems is undecidable.*
(2) *Let $\mathcal{F}$ be specifications of sets of finite executions expressed as NFW or LTLf formulas. Then $PMCP(\mathcal{F})$ of TNC-Systems is decidable.*

5. Solving PMCP for Specifications over Finite Executions

In this section we solve the PMCP problem for specifications given as nondeterministic finite word automata (NFW), and prove that it is PSPACE-complete. Following the automata-theoretic approach outlined in Section 3.4, given an RB-template P we will build an NFW

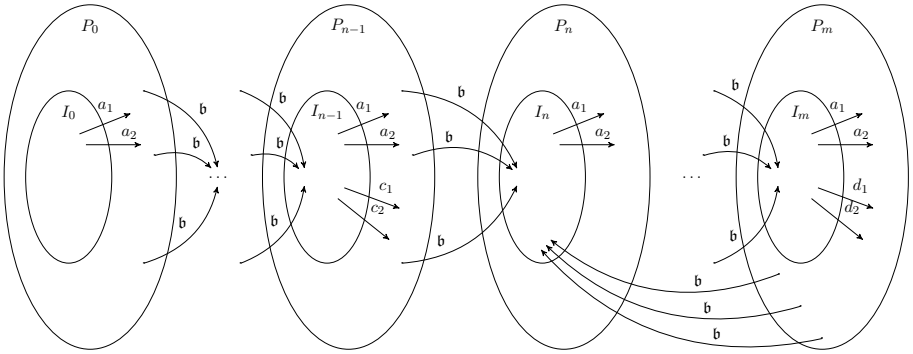


Figure 5: A high level view of the reachability-unwinding lasso.

$\mathcal{A}$ that accepts exactly the executions in EXEC-FIN(P^∞). Model checking of a regular specification given by an NFW $\mathcal{A}'$ is thus reduced to checking containment of the language of $\mathcal{A}$ in that of $\mathcal{A}'$. The structure of $\mathcal{A}$ is based on the reachability-unwinding of P, which we now introduce.

Given a template $P = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S, I, R, \lambda\rangle$, we show how to construct a new process template $P^{\multimap} = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S^{\multimap}, I^{\multimap}, R^{\multimap}, \lambda^{\multimap}\rangle$, called the *reachability-unwinding* of P, see Figure 5. Intuitively, $P^{\multimap}$ is obtained by alternating the following two operations: *(i)* taking a copy of P and removing from it all the unreachable rendezvous edges, i.e., all transitions of P that cannot be taken by any process in any run of P^∞; 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.

Intuition. Technically, it is more convenient to first calculate all the desired copies and then to arrange them in the lasso. Thus, for $0 \leq i \leq m$ (for an appropriate m), we first compute an R-template $P_i = \langle AP, \Sigma_{\mathsf{rdz}}, S_i, I_i, R_i, \lambda_i\rangle$ 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 $P^{\multimap}$. We do this by connecting, for $i < m$, the states in P_i with the initial states of P_{i+1} by means of broadcast edges, as induced by transitions in P. In case $i = m$, then P_i is connected to the copy P_n, for some $n \leq m$, as determined by the lasso structure.

Constructing P_i via a Saturation Algorithm. Recursively construct the R-template

$$P_i = \langle AP, \Sigma_{\mathsf{rdz}}, S_i, I_i, R_i, \lambda_i\rangle$$

(called the i'th *component* of $P^{\multimap}$) as follows. For $i = 0$, we let $I_0 := I$; and for $i > 0$ we let $I_i := \{s \in S \mid (h, \mathfrak{b}, s) \in R \text{ for some } h \in S_{i-1}\}$ be the set of states reachable from S_{i-1} by a broadcast edge. The sets S_i and R_i are obtained using the following *saturation* algorithm: start with $S_i := I_i$ and $R_i := \emptyset$; at each round of the algorithm, consider in turn each edge $e \in R \setminus R_i$ of the form $s \xrightarrow{a_h} t$ such that $s \in S_i$: if for every $l \in [k] \setminus \{h\}$ there is some other edge $s' \xrightarrow{a_l} t'$ 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. Finally, let λ_i be the restriction of λ to $S_i \times AP$. Observe the following property of this algorithm:

Remark 5.1. If $s \xrightarrow{a_h} t$ in R_i then for all $l \in [k]$ there exist $s', t' \in S_i$ such that $s' \xrightarrow{a_l} t'$ 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, there must exist n, m with $n \leq m < 2^{|S|}$ such that $P_n = P_{m+1}$. We stop calculating P_i's when this happens since for every $i \geq n$ it must be that $P_i = P_{n+((i-n) \mod r)}$, where $r = m+1-n$. We call n the *prefix length* of $P^{-\circ}$ and call r its *period*, i.e., n is the number of components on the prefix of the lasso and r is the number of components on the noose of the lasso. For $i \in \mathbb{N}$, let $comp(i)$ denote the associated component number of i, i.e., the position in the lasso after i moves between components. Formally, $comp(i) = \min(i, n + ((i-n) \mod r))$.

We now construct $P^{-\circ}$.

Definition 5.2 (Reachability-unwinding)**.** Given $P_0, \ldots, P_m$, define the RB-template

$$P^{-\circ} = \left\langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S^{-\circ}, I^{-\circ}, R^{-\circ}, \lambda^{-\circ} \right\rangle$$

as follows:

- $S^{-\circ} := \cup_{i=0}^{m} \{(s,i) \mid s \in S_i\}$;
- $I^{-\circ} := \{(s,0) \mid s \in I\}$;
- $R^{-\circ}$ contains the following transitions:
  - the rendezvous transitions $\cup_{i=0}^{m} \{(s,i) \xrightarrow{\varsigma} (t,i) \mid s \xrightarrow{\varsigma} t \in R_i\}$, and
  - the broadcast transitions $\cup_{i=0}^{m-1} \{(s,i) \xrightarrow{\mathfrak{b}} (t,i+1) \mid s \xrightarrow{\mathfrak{b}} t \in R \text{ and } s \in S_i\}$ and $\{(s,m) \xrightarrow{\mathfrak{b}} (t,n) \mid s \xrightarrow{\mathfrak{b}} t \in R \text{ and } s \in S_m\}$.
- $\lambda^{-\circ} = \cup_{i=0}^{m} \{((s,i),p) : (s,p) \in \lambda_i\}$.

For $0 \leq i \leq m$, we denote by $P_i^{-\circ}$ the restriction of $P^{-\circ}$ to the states in $\{(s,i) \mid s \in S_i\}$ and the rendezvous transitions between them. We call $P_i^{-\circ}$ the *i'th component of* $P^{-\circ}$. Observe that $P_i^{-\circ}$ can be written as an R-template $P_i^{-\circ} = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S_i^{-\circ}, I_i^{-\circ}, R_i^{-\circ}, \lambda_i^{-\circ} \rangle$ which is obtainable from the component P_i by simply attaching i to every state. We will sometimes find it convenient to speak of the component $P_i^{-\circ}$, for i greater than m, in which case we identify $P_i^{-\circ}$ with $P_{comp(i)}^{-\circ}$. We say that a configuration f of $(P^{-\circ})^{\infty}$ is *in* $P_i^{-\circ}$ iff all its processes are in states of the component $P_i^{-\circ}$, i.e., iff $f(j) \in S_i^{-\circ}$ for all j in the domain of f.

Example 5.3.

- If P is the template in Figure 1 then $P_0 = P$, and P_1 is the empty process (because there are no broadcast edges). Thus $P^{-\circ}$ is a copy of P.
- If P is the template in Figure 2 then P_0 is equal to P without the broadcast edges, and $P_1 = P_0$. Thus $P^{-\circ}$ is a copy of P.
- If P is the template on the right hand side of Figure 4, then $P^{-\circ}$ contains four components (prefix length 3 and period 1). The four components are drawn in Figure 6.

Definition 5.4 (Legal configuration/path)**.** A configuration f of $(P^{-\circ})^{\infty}$ is *legal* iff it is in $P_i^{-\circ}$ for some i; a path in $(P^{-\circ})^{\infty}$ is *legal* iff its source configuration is.

Remark 5.5. If π is a finite path of $(P^{-\circ})^{\infty}$, with b broadcast transitions, with source f and destination f', then if f is in $P_i^{-\circ}$ then f' is in $P_{i+b}^{-\circ}$, and if π is a run then f' is in $P_b^{-\circ}$. In particular, any configuration of $(P^{-\circ})^{\infty}$ that is reachable from an initial configuration is legal.

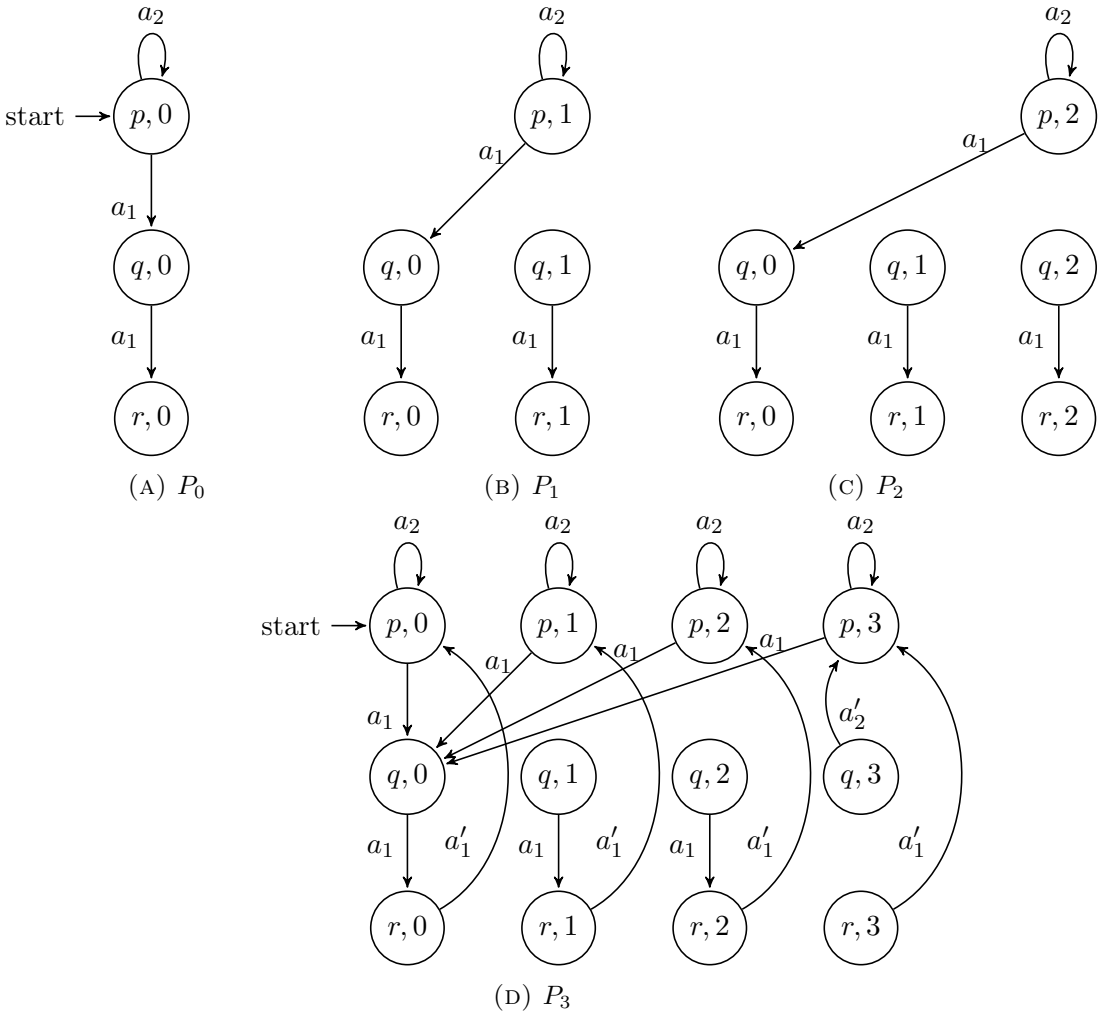


Figure 6: Components P_0, P_1, P_2, P_3 of $P^{-\circ}$ for P from Figure 4.

Recall that we introduced $P^{-\circ}$ in order to define an automaton recognizing EXEC-FIN(P^∞). Before doing so, we have to understand the relationship between $P^{-\circ}$ and P.

On the relation between $P^{-\circ}$ and P. Observe that by projecting out the component numbers from states in $P^{-\circ}$ (i.e., by replacing $(s,i) \in S^{-\circ}$ with $s \in S$), states and transitions in $P^{-\circ}$ are transformed into states and transitions of P. Similarly, paths and runs in $(P^{-\circ})^\infty$ can be transformed into paths and runs in P^∞. Note, however, that this operation does not induce a bisimulation between $P^{-\circ}$ and P, nor does it induce a bisimulation relation between P^∞ and $(P^{-\circ})^\infty$, since not all states and transitions of P appear in every component of $P^{-\circ}$. These missing states and edges are also the reason that a path in 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 $(P^{-\circ})^\infty$. Nonetheless, our construction of $P^{-\circ}$ is such that runs of P (resp. P^∞) can be lifted to runs of $P^{-\circ}$ (resp. $(P^{-\circ})^\infty$) by simply adding the correct component numbers (based on the number of preceding broadcasts) to the states of the transitions of the run.

Winding and Unwinding Notation. More formally, projecting out of component numbers (which we call "winding" and denote by $\circledcirc$) is defined as follows: if (s,j) is a state of $P^{-\circ}$ define $(s,j)^{\circledcirc} := s$, which is a state of P; if t is a transition $s \xrightarrow{\varsigma} s'$ of $P^{-\circ}$ define $t^{\circledcirc} := (s^{\circledcirc}, \varsigma, s'^{\circledcirc})$, which is a transition of P; if $\pi = t_1 t_2 \cdots \in \mathit{runs}(P^{-\circ})$ define $\pi^{\circledcirc} := t_1^{\circledcirc} t_2^{\circledcirc} \cdots \in \mathit{runs}(P)$. Similarly, if $\mathfrak{f}$ is a configuration in $(P^{-\circ})^\infty$ define $\mathfrak{f}^{\circledcirc}$, a configuration of P^∞, by $\mathfrak{f}^{\circledcirc}(i) := \mathfrak{f}(i)^{\circledcirc}$ where i is in the domain of $\mathfrak{f}$; if e is a global transition $\mathfrak{f} \xrightarrow{\sigma} \mathfrak{g}$ of $(P^{-\circ})^\infty$ then define $e^{\circledcirc} := (\mathfrak{f}^{\circledcirc}, \sigma, \mathfrak{g}^{\circledcirc})$; and if $\rho \in \mathit{runs}((P^{-\circ})^\infty)$ define $\rho^{\circledcirc} = \rho_1^{\circledcirc} \rho_2^{\circledcirc} \cdots \in \mathit{runs}(P^\infty)$. Finally, we apply this to sets: if $X \subseteq \mathit{runs}((P^{-\circ})^\infty)$ then $X^{\circledcirc} = \{\rho^{\circledcirc} : \rho \in X\}$.

We define the reverse transformation of "unwinding" only with respect to runs of P^∞ (a similar definition can be given for the unwinding of runs of P) as follows: given a configuration f of P^∞, and a component number j, denote by f^j the function defined by $f^j(i) := (f(i), j)$ for every i in the domain of f; given $\pi \in \mathit{runs}(P^\infty)$, for $i \in \mathbb{N}_{>0}$ let $\mathfrak{b}^{<i}$ be the number of broadcast transitions on π preceding π_i. The *unwinding* $\pi^{-\circ}$ of π is defined to be the sequence $\pi_1^{-\circ}\pi_2^{-\circ}\ldots$ obtained by taking for every $1 \le i \le |\pi|$ the transition $\pi_i^{-\circ} := (f^{\mathit{comp}(\mathfrak{b}^{<i})}, \sigma, g^{\mathit{comp}(\mathfrak{b}^{<i})})$ if $\pi_i = (f, \sigma, g)$ is a rendezvous transition, and otherwise taking $\pi_i^{-\circ} := (f^{\mathit{comp}(\mathfrak{b}^{<i})}, \mathfrak{b}, g^{\mathit{comp}(\mathfrak{b}^{<i+1})})$ if $\pi_i = (f, \mathfrak{b}, g)$ is a broadcast transition.

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

Lemma 5.6. *For every $n \in \mathbb{N}_{>0}$, we have that $\mathit{runs}(P^n) = \mathit{runs}((P^{-\circ})^n)^{\circledcirc}$.*

Proof. Let us fix any $n \in \mathbb{N}_{>0}$. The direction $\{\rho^{\circledcirc} \mid \rho \in \mathit{runs}((P^{-\circ})^n)\} \subseteq \mathit{runs}(P^n)$ follows from the fact that $P^{-\circ}$ is obtained from P by an unwinding process. The reverse inclusion requires more care as $P^{-\circ}$ misses edges and states of P.

Let $\pi \in \mathit{runs}(P^n)$. We prove that $\pi^{-\circ} \in \mathit{runs}((P^{-\circ})^n)$ by induction on the length i of each prefix of π. Let b be the number of broadcast edges on $\xi := \pi_1 \ldots \pi_{i-1}$. For the base case $|\pi| = 0$, there is nothing to prove. For the induction step, observe that by the inductive hypothesis the unwinding of ξ is a run of $(P^{-\circ})^n$. It remains to show that $\pi_i^{-\circ}$ is a transition of $(P^{-\circ})^n$. Observe that by Remark 5.5 $f = \mathsf{dst}(\pi_{i-1}) = \mathsf{src}(\pi_i)$ is in $P^{-\circ}_{\mathit{comp}(b)}$. Consider first the case that π_i is the broadcast transition $f \xrightarrow{\mathfrak{b}} g$. By the definition of the broadcast edges in $P^{-\circ}$ we have that $\pi_i^{-\circ}$ is a transition $f^{\mathit{comp}(b)} \xrightarrow{\mathfrak{b}} g^{\mathit{comp}(b+1)}$ of $(P^{-\circ})^n$. Consider now the case that π_i is the rendezvous edge $f \xrightarrow{\sigma} g$, and let $\sigma = ((j_1, a_1), \ldots, (j_k, a_k))$. Since f is in $P^{-\circ}_{\mathit{comp}(b)}$, for every $h \in [k]$, the algorithm used to construct the states and transitions of the component $P_{\mathit{comp}(b)}$ must have added the edge $f(j_h) \xrightarrow{a_h} g(j_h)$ to $R_{\mathit{comp}(b)}$. It follows that $\pi_i^{-\circ} = (f^{\mathit{comp}(b)}, \sigma, g^{\mathit{comp}(b)})$ is a transition of $(P^{-\circ})^n$. □

The state labeling of a run $\rho \in \mathit{runs}((P^{-\circ})^\infty)$ and its winding $\rho^{\circledcirc}$ are equal. Thus we have the following:

Corollary 5.7. *For every template P, we have that $\textsc{exec}(P^\infty) = \textsc{exec}((P^{-\circ})^\infty)$.*

The following lemma says that for every component $P_b^{-\circ}$, there is a run of $(P^{-\circ})^\infty$ that "loads" arbitrarily many processes into every state of it.

Lemma 5.8 (Loading)**.** *For all $b, n \in \mathbb{N}_{>0}$ there is a finite run π of $(P^{-\circ})^\infty$ with b broadcasts, s.t., $|\mathsf{dst}(\pi)^{-1}(s)| \ge n$ for every state s of $P_b^{-\circ}$.*

Proof. By Lemma 3.12 (Composition) applied to $P^{-\circ}$ it is sufficient to prove the following: for every $b \in \mathbb{N}_{>0}$, and every state q in $P_b^{-\circ}$, there exists a finite run π of $(P^{-\circ})^\infty$, with b broadcast transitions, such that $|\mathsf{dst}(\pi)^{-1}(q)| \ge 1$. Recall that, by definition, $P_b^{-\circ} = P^{-\circ}_{\mathit{comp}(b)}$.

The proof is by induction on b. For the base case $b = 0$, proceed by induction on the round $j \geq 1$ of the saturation algorithm at which q is added to $S_{comp(b)}$ (i.e., S_0). The state q is added at round j, due to some edge (s_h, a_h, q) of R, only if for every $l \in [k] \setminus \{h\}$ there are edges (s_l, a_l, q_l) of R and, either (i) $j = 1$ and $s_l \in I_{comp(b)}$ or, (ii) $j > 1$ and s_l is already in $S_{comp(b)}$ (i.e., it was added to $S_{comp(b)}$ at a round before j). By the inductive hypothesis on round j, for every $l \in [k] \setminus \{h\}$ there exists $\rho_l \in runs((P^{-\circ})^\infty)$, with b broadcasts, which ends with at least one process in the state s_l. By Lemma 3.12 (Composition) there exists $\rho \in runs((P^{-\circ})^\infty)$ in which there are k different processes $i_1, \ldots, i_k$ such that, for every $l \in [k]$, the process i_l ends in the state s_l. Extend ρ by a global rendezvous transition in which, for $l \neq h$, process i_l takes the edge (s_l, a_l, q_l), and process i_h takes the edge (s_h, a_h, q). This extended run has b broadcast transitions, and at least one process in state q, as required.

For the inductive step ($b > 0$), suppose it holds for all values $\leq b$, and let us prove it for $b + 1$ (i.e. take $q \in S_{comp(b+1)}$). First consider the case $q \in I_{comp(b+1)}$: there is an edge $(s, \mathfrak{b}, q)$ in $P^{-\circ}$ and by the inductive hypothesis (on b) there is a run of $(P^{-\circ})^\infty$ with b broadcasts in which some process i ends in state s. Extend this run by a global broadcast transition in which process i takes the edge $(s, \mathfrak{b}, q)$. This extended run has $b + 1$ broadcast transitions, and at least one process in state q. Second, suppose $q \in S_{comp(b+1)} \setminus I_{comp(b+1)}$. Then proceed as in the base case. □

The first part of the following proposition states that the set of finite executions of the RB-system P^∞ is equal to the set of state labels of the finite runs of $P^{-\circ}$. This is very convenient since $P^{-\circ}$ is finite, whereas P^∞ is infinite. The second part of the proposition gives a weaker result for the infinite case.

Proposition 5.9. *For every template P, the following holds:*

(1) $\textsc{exec-fin}(P^\infty) = \{\lambda^{-\circ}(\pi) \mid \pi \in runs(P^{-\circ}), |\pi| \in \mathbb{N}_{>0}\}$.
(2) $\textsc{exec-inf}(P^\infty) \subseteq \{\lambda^{-\circ}(\pi) \mid \pi \in runs(P^{-\circ}), |\pi| = \infty\}$.

Proof. We first prove the inclusion $\textsc{exec}(P^\infty) \subseteq \{\lambda^{-\circ}(\pi) \mid \pi \in runs(P^{-\circ})\}$. Every execution of P^∞ is, by definition, of the form $\lambda(proj_\xi(1))$ for some $\xi \in runs(P^n)$ and some n. By Lemma 5.6, $\xi = \rho^\circledcirc$ for some $\rho \in runs((P^{-\circ})^n)$. Observe that ξ and ρ are equi-labeled. Thus, by Lemma 3.5 part 1 we have that $\lambda(proj_\xi(1)) = \lambda^{-\circ}(proj_\rho(1))$.

We now prove the inclusion

$$\{\lambda^{-\circ}(\pi) \mid \pi \in runs(P^{-\circ}), |\pi| \in \mathbb{N}_{>0}\} \subseteq \textsc{exec-fin}(P^\infty).$$

Observe that since $\lambda^{-\circ}(\pi) = \lambda(\pi^\circledcirc)$ it is enough to prove the following by induction on the length i of π: there is a run $\rho \in runs(P^\infty)$ such that $proj_\rho(1) = \pi^\circledcirc$.

For the base case $i = 0$ there is nothing to prove. For the inductive step $i > 0$: first apply the inductive hypothesis to get $\rho \in runs(P^\infty)$ such that $proj_\rho(1) = (\pi_1 \pi_2 \cdots \pi_{i-1})^\circledcirc$. There are two cases depending on π_i.

If π_i is a broadcast edge then extend ρ by a global broadcast transition t in which process 1 takes $\pi_i^\circledcirc$, i.e., $edge_1(t) = \pi_i^\circledcirc$, to obtain the run $\rho \cdot t \in runs(P^\infty)$ whose projection on process 1 equals $\pi^\circledcirc$.

If $\pi_i = (s, a_h, t)$ is a rendezvous edge then proceed as follows. Let b be the number of broadcast transitions in $\pi_1 \cdots \pi_{i-1}$. So π_i, being a rendezvous edge, is in $P_b^{-\circ}$. Thus, by Remark 5.1, after the saturation algorithm, for all $l \in [k]$ there exist an edge (s_l, a_l, t_l) in $P_b^{-\circ}$. By Lemma 5.8 (Loading) there exists $\rho' \in runs((P^{-\circ})^\infty)$ with b broadcast transitions that loads at least one process into every state s of $P_b^{-\circ}$. By Lemma 5.6, $\rho'^\circledcirc \in runs(P^\infty)$.

By Lemma 3.12 (Composition) compose ρ and $\rho'^{\circledcirc}$ to get $\rho'' \in \mathit{runs}(P^\infty)$ such that $\rho''\big|_{\{1\}} = \mathit{proj}_\rho(1) = (\pi_1\pi_2\cdots\pi_{i-1})^{\circledcirc}$ and at the end of ρ'' there is at least one process (different from process 1) in every state of $P_b^{-\circ}$. Now extend ρ'' by the rendezvous transition t for which $\mathit{edge}_1(t) = \pi_i^{\circledcirc}$ and for each $l \in [k] \setminus \{h\}$ some process takes the transition (s_l, a_l, t_l) to obtain the run $\rho \cdot t \in \mathit{runs}(P^\infty)$ whose projection on process 1 equals $\pi^{\circledcirc}$. □

Remark 5.10. Unfortunately, the containment in Proposition 5.9 Part 2 is sometimes strict. For example, consider the R-template P in Figure 1. Observe that P equals $P^{-\circ}$, and that p^ω is the state label of the run of $P^{-\circ}$ that self-loops in the initial state forever, but p^ω is not an execution of P^∞. In the next section we will use B-automata to capture EXEC-INF(P^∞).

We introduced $P^{-\circ}$ in order to define an automaton recognizing EXEC-FIN(P^∞). This automaton is formed from the LTS $P^{-\circ}$ by adding the input alphabet 2^{AP} and having each transition read as input the label of its source state.

Definition 5.11 (NFW $\mathcal{A}$)**.** Given an RB-template $P = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S, I, R, \lambda\rangle$, consider the reachability-unwinding $P^{-\circ} = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S^{-\circ}, I^{-\circ}, R^{-\circ}, \lambda^{-\circ}\rangle$. Define $\mathcal{A}$ to be the NFW $\langle \Sigma, S', I', R', F\rangle$ with

- input alphabet $\Sigma = 2^{AP}$,
- state set $S' = S^{-\circ}$,
- initial-states set $I' = I^{-\circ}$,
- transition relation R' consisting of transitions $(s, \lambda^{-\circ}(s), t)$ for which there is a σ such that $(s, \sigma, t) \in R^{-\circ}$,
- final-states set $F = S^{-\circ}$.

The following is immediate from Proposition 5.9 Part 1:

Corollary 5.12. *The automaton $\mathcal{A}$ recognizes the language* EXEC-FIN(P^∞).

Applying a standard automata-theoretic technique, we get the following upper bound:

Theorem 5.13. *Let $\mathcal{F}$ be specifications of finite executions expressed as NFWs or LTLf formulas. Then $PMCP(\mathcal{F})$ for RB-systems is in* PSPACE.

Proof. Let $P = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S, I, R, \lambda\rangle$ be a process template, and let $\mathcal{A}$ be the NFW from Definition 5.11. The fact that words accepted by $\mathcal{A}$ are exactly the executions in EXEC-FIN(P^∞) is by Corollary 5.12. Analyzing the construction of the unwinding template $P^{-\circ}$ (before Definition 5.2), we get that $\mathcal{A}$ is of size at most exponential in the size of P.

We describe a PSPACE algorithm for checking the containment of the language accepted by $\mathcal{A}$ in the language of some specification NFW $\mathcal{A}'$. This is done by solving the non-containment problem in nondeterministic polynomial space, and using the fact that NPSPACE = PSPACE = co-PSPACE.[7] The algorithm constructs on the fly: (1) a finite word $\rho \in (2^{AP})^*$, and an accepting run of $\mathcal{A}$ on ρ; and (2) checks that ρ is not accepted by $\mathcal{A}'$. Item (2) can be done, as usual, simply by storing the subset of states of $\mathcal{A}'$ that are reachable by reading the prefix of ρ constructed thus far, and validating that, at the end, this set does not contain an accepting state. For item (1), the algorithm does not store all of (the exponentially large $\mathcal{A}$). Instead, at each point in time, it only stores a single component $P_i^{-\circ}$ of $P^{-\circ}$ (which can

[7] Our nondeterministic algorithm will be allowed to diverge. Such an algorithm can be simulated by one that never diverges and still runs in polynomial space by incrementing a counter at every step, and rejecting the computation-branch if the counter ever exceeds the original number of configurations of the possibly-diverging algorithm.

be calculated in PTIME for every i), from which it can deduce the part of $\mathcal{A}$ corresponding to it. The algorithm starts by constructing $P_0^{\circlearrowright}$, and sets ρ to be the empty word, and the run of $\mathcal{A}$ on ρ to be the initial state of $\mathcal{A}$. At each step, it can either declare the guess as finished (if the run constructed thus far ends in an accepting state of $\mathcal{A}$) or extend ρ and the run. Extending ρ is a trivial guess. Extending the guessed run is done by either guessing a transition of $\mathcal{A}$ inside the component induced by the currently stored $P_i^{\circlearrowright}$; or by guessing a transition that moves to the next component in the lasso, at which point the algorithm also discards $P_i^{\circlearrowright}$ and replaces it with $P_{i+1}^{\circlearrowright}$.

In case the specification is given as an LTLf formula φ, we let $\mathcal{A}'$ be the NFW from Theorem 2.2 corresponding to $\neg\varphi$ and replace (2) above by a check that ρ is accepted by $\mathcal{A}'$, which can be done, as usual, simply by storing the subset of states of $\mathcal{A}'$ that are reachable by reading the prefix of ρ constructed thus far, and validating that, at the end, this set does contain an accepting state. □

The next theorem gives a corresponding lower bound. Interestingly, its proof shows that the problem is already PSPACE hard for *safety* specifications (i.e., that a bad state is never visited).

Theorem 5.14. *Let $\mathcal{F}$ be specifications of finite executions expressed as NFW or LTLf formulas. Then $PMCP(\mathcal{F})$ for RB-systems is PSPACE-hard. Moreover, this is true even for a fixed specification, and thus the program-complexity is PSPACE-hard.*

Proof. The proof proceeds by a reduction from the reachability problem for Boolean programs, known to be PSPACE-complete [Jon97].

A Boolean program consists of m Boolean variables $X_1, \ldots, X_m$ (for some m) and n instructions (for some n), referred to by their *program location* $l \in [n]$, of two types: *(i) conditionals* of the form $l : \texttt{if } X_i \texttt{ then } l_{if} \texttt{ else } l_{else}$; *(ii) toggles* of the form $l : X_i := \neg X_i$. The semantics of the first type of instruction is to move from location l to location l_{if} if X_i is true and to location l_{else} otherwise; instructions of this type are thus conditional jumps that do not change the values of any of the Boolean variables. The semantics of the second type of instruction is to negate the value of the Boolean variable X_i; the execution continues from location $l+1$ (unless that was the last instruction). All Boolean variables are initialized to `false` and execution begins with instruction 1. We remark that the Boolean programs considered here are deterministic. The reachability problem for Boolean programs is to decide whether the execution of a Boolean program ever reaches its last program location n. Note that we can assume, without loss of generality, that the last instruction of a Boolean program is a conditional instruction.

Given a Boolean program B, we build a process template P, and a specification NFW, such that P^∞ satisfies the specification iff the execution of B does not reach its last instruction.

Formally, $P = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S, \{\iota\}, R, \lambda\rangle$, where:

- $AP = \{\text{done}\}$, i.e., there is a single atom;
- $\Sigma_{\mathsf{rdz}} = \cup_{a\in\Sigma_{\mathsf{actn}}}\{a_1, a_2\}$ where
$$\Sigma_{\mathsf{actn}} = \cup_{i\in[m]}\{(\text{protect}, i), (\text{if}, i), (\text{else}, i), (\text{toggle}, i)\}$$
- The set of states $S := \{\iota, \texttt{sink}\} \cup S_{instr} \cup S_{var}$, where ι is an initial state, `sink` is a sink state, and:
  (1) $S_{instr} := \cup_{l\in[n]}\{l, l'\}$,
  (2) $S_{var} := \cup_{i\in[m]}\{X_i, \neg X_i, X_i', \neg X_i'\}$,
- R will be defined later;

- $\lambda = \{(n, \text{done})\}$, i.e., the atom is true in state n, and false in all other states.

The specification says that the last program location n is never visited. This can be expressed, for instance, by the LTLf formula $\mathsf{G}\,\neg\text{done}$ (read "it is always the case that atom done does not occur"), and by an NFW consisting of a single state. In what follows we call the specification φ.

Before describing the transition relation R, we briefly describe the way the states are used in the simulation of the Boolean program B by runs of P^∞. At the beginning of every run, every process nondeterministically moves (on a broadcast) from the initial state either to the state 1, or to one of the states $\neg X_1, \cdots, \neg X_m$. A process that moves to the state 1 will keep track of (i.e., encode) the program location of the Boolean program, and from this point on will only be in states from the set S_{instr}; whereas a process that moves to a state of the form $\neg X_i$ will keep track of (i.e., encode) the value of variable X_i, and from this point on will only be in states from the set $\{X_i, \neg X_i, X_i', \neg X_i'\}$. Observe that multiple processes may decide to encode the program location or the same variable. However, the transition relation R will be defined in such a way as to enforce the invariant that, right after every broadcast, the following holds:

(†) : all processes that encode the same object (i.e., the program location or the value of some variable) agree on its value.

Moreover, between every two broadcasts, at most one instruction of the Boolean program can be simulated, namely, the instruction referenced by the processes that track the program location. The primed versions of the states will be used in order to enforce this round structure, as well as the invariant †, as follows: in each round, every rendezvous transition moves a process from an unprimed state to a primed state, from which it can only move on a broadcast; whereas a broadcast takes a process in a primed state back to an unprimed state, and processes in an unprimed state to `sink`.

Let $var(l)$ denote the index i of the variable used (i.e., tested or toggled) in the instruction in program location l. We now define the transition function R of the template P. It consists of the following transitions:

- $\iota \xrightarrow{\mathfrak{b}} 1$; $\iota \xrightarrow{\mathfrak{b}} \neg X_i$ for $i \in [m]$,
- $\texttt{sink} \xrightarrow{\mathfrak{b}} \texttt{sink}$; $l \xrightarrow{\mathfrak{b}} \texttt{sink}$, $X_i \xrightarrow{\mathfrak{b}} \texttt{sink}$, $\neg X_i \xrightarrow{\mathfrak{b}} \texttt{sink}$, for $l \in [n], i \in [m]$,
- $l' \xrightarrow{\mathfrak{b}} l$, $X_i' \xrightarrow{\mathfrak{b}} X_i$, and $\neg X_i' \xrightarrow{\mathfrak{b}} \neg X_i$.
- $l \xrightarrow{(\text{protect},i)_1} l$ for $l \in [n]$ and $i \in [n] \setminus var(l)$.
- $X_i \xrightarrow{(\text{protect},i)_2} X_i'$ and $\neg X_i \xrightarrow{(\text{protect},i)_2} \neg X_i'$ for $i \in [m]$.
- $l \xrightarrow{(\text{if},var(l))_1} l'_{if}$ and $l \xrightarrow{(\text{else},var(l))_1} l'_{else}$ for all conditional instructions l.
- $X_i \xrightarrow{(\text{if},i)_2} X_i'$ and $\neg X_i \xrightarrow{(\text{else},i)_2} \neg X_i'$ for $i \in [m]$.
- $l \xrightarrow{(\text{toggle},var(l))_1} (l+1)'$ for all toggle instructions l.
- $X_i \xrightarrow{(\text{toggle},i)_2} \neg X_i'$ and $\neg X_i \xrightarrow{(\text{toggle},i)_2} X_i'$ for $i \in [m]$.

We now prove that the reduction is correct.

Suppose that the infinite run ρ of the Boolean program visits its last instruction. We build a run π of P^{m+1} witnessing the fact that P^∞ does not satisfy φ. The run π simulates ρ as follows. Start with a broadcast, which takes process $m+1$ (called the *controller process*) to state 1, and for every $i \in [m]$ takes process i (called the *i'th variable process*) to $\neg X_i$.

Repeatedly extend the run π by the following sequence of global transitions (below, l denotes the current state of the controller process):

(1) For every $i \neq var(l)$, the controller rendezvous with the i'th memory process on the action $(\text{protect}, i)$.
(2) The controller rendezvous with the $var(l)$'th memory process as follows: if l is a toggle instruction then the rendezvous action is (toggle, i); otherwise, it is (if, i) if the i'th memory process is in state X_i, and it is (else, i) if it is in state $\neg X_i$.
(3) A broadcast.

It is easy to see that π simulates ρ. In particular, the state of the controller after $z \geq 1$ broadcasts is equal to the program location of the Boolean program after z steps.

For the other direction, we argue as follows. We say that a configuration f of P^∞ is *consistent* if it satisfies the invariant † stated earlier. For such an f, let $pl(f) \in [n]$ be the program location encoded by f, or $\perp$ if there are no processes in f tracking the program location; and for every $i \in [m]$, let $val_i(f) \in \{\texttt{true}, \texttt{false}\}$ be the value of X_i encoded by f, or $\perp$ if there are no processes in f tracking the value of X_i. Given $z \in \mathbb{N}$, and a run π of P^∞ with at least z broadcasts, write $\pi(z)$ for the configuration in π immediately following the z'th broadcast. Observe that it is enough to show that π simulates the run ρ of the Boolean program in the following sense:

(1) $\pi(z)$ is consistent,
(2) if $pl(f_z) \neq \perp$ then the program location in ρ_z is equal to $pl(f_z)$,
(3) for every $i \in [m]$, if $val_i(f_z) \neq \perp$ then the value of variable X_i in ρ_z is equal to $val_i(f_z)$.

We prove the items above by induction on z. For $z = 1$, i.e., after the first broadcast (which must be the first transition on any run), processes assume different roles. Any process that moves to state 1 is called a *controller*, and any process that moves to state $\neg X_i$ (for some $i \in [m]$) is called an *i'th variable processes*. Clearly the induction hypothesis holds. For the inductive step, note that by the inductive hypothesis $\pi(z-1)$ is consistent, let $l := pl(\pi(z-1))$, and observe that the only rendezvous transitions on π between the $z-1$ and z broadcasts are of a controller process that rendezvous with a variable process on an action of the form described in items 1 and 2 in the proof of the first direction (in particular, if $l = \perp$ then there are no rendezvous between the $z-1$ and z broadcasts). Thus, it must be that, just before the z'th broadcast, processes in a primed state that are encoding the same object are in the same state. Combining this with the fact that any process in an unprimed state will move to `sink` on the z'th broadcast one can see that the inductive hypothesis holds also after the z'th broadcast. □

Remark 5.15. We now show that specification complexity of the PMCP for NFW and LTLf specifications is also PSPACE-hard. We do this by reducing from the standard model-checking problem.

Recall that the standard model-checking problem is, given an LTS L without edge labels (aka 'finite state program' or 'Kripke structure') and a specification φ, to decide if all finite executions of L satisfy φ. The specification complexity of the model-checking problem for LTLf formulas or for NFW specifications is PSPACE-hard. To see that, given an alphabet Σ, take a single state Kripke structure K that generates all words in Σ^*, and note that model-checking K and a given NFW specification is equivalent to deciding the universality problem for NFWs which is (even over a fixed alphabet) PSPACE-hard [GJ79]. Similarly, model-checking K and a given LTLf formula is equivalent to deciding LTLf satisfiability

(using the negation of the original formula) which is again PSPACE-hard even for a fixed alphabet P [DGV13].

To reduce the model-checking problem (for a fixed LTS L without edge labels) to the PMCP problem with a fixed RB-template, simply build an RB-template P from L by adding the edge label $\mathfrak{b}$ to every transition, i.e., every transition in L becomes a broadcast transition. Clearly, then, EXEC-FIN(P^∞) is exactly the sequences of the form $\lambda(\pi)$ where π is a finite run of L. Thus, $L \models \varphi$ iff all executions in EXEC-FIN(P^∞) satisfy φ.

Theorem 3.7 from Section 3.4 now follows: the upper bound is in Theorem 5.13, the lower-bound on the program complexity (and thus the combined complexity) is in Theorem 5.14, and the lower-bound on the specification complexity is in Remark 5.15.

6. Solving PMCP for Specifications over Infinite Executions

The main step in our automata-theoretic approach to solve the PMCP for infinite executions is the construction, given an RB-template P, of a B-automaton $\mathcal{B}$ (with a trivial Büchi set) that accepts the language EXEC-INF(P^∞). In this section we describe the construction of this automaton.

In order to construct the B-automaton $\mathcal{B}$, it is helpful to recall the source of difficulty in dealing with infinite executions as opposed to finite executions. Recall from Section 5 that the finite executions were dealt with by simply turning the reachability-unwinding $P^{-\circ}$ into a nondeterministic automaton $\mathcal{A}$ (by having each transition read as input the label of its source state), and that this worked because of the equality between the state-labels of finite runs of $P^{-\circ}$ and the finite executions of P^∞, as stated in the first part of Proposition 5.9. Also, recall that the second part of the same proposition, which deals with the infinite case, only states a containment (instead of equality), which may be strict — as illustrated by the template P in Example 3.3. Indeed, looking at this template again, one can see that in order to allow process 1 to trace p^z for $z \in \mathbb{N}_{>0}$, we can use a system with $z+1$ processes that rendezvous with process 1 one after the other. However, no finite amount of processes can allow process 1 to trace p^ω, since once a process rendezvous with process 1 it cannot do so ever again. Thus, while the self loop on the initial state can be taken infinitely often in a path in the template P (and hence also in a run of the automaton $\mathcal{A}$), it can not be taken infinitely often in a run of P^∞.

The key to modifying $\mathcal{A}$ to obtain $\mathcal{B}$ is to treat edges of $P^{-\circ}$ differently based on the conditions under which they can (or cannot) be taken infinitely often in runs of $(P^{-\circ})^\infty$. In particular, one has to distinguish between edges that can appear infinitely often on a run with finitely or infinitely many broadcasts, and among the latter between ones that can or cannot appear unboundedly many times between two consecutive broadcasts. Note that the fact that an edge is only taken a bounded number of times between consecutive broadcasts can be naturally tracked by the acceptance condition of a single counter.

The rest of this section is organized as follows. We formally present the classification of edges along the lines outlined above, and prove a couple of easy lemmas about this classification. We then give the definition of the automaton $\mathcal{B}$ and prove the correctness of the construction.

Definition 6.1 (Edge Types)**.** An edge e of $P^{-\circ}$ is

- *locally-reusable* iff it appears infinitely many times on some run of $(P^{-\circ})^\infty$ with finitely many broadcasts.

- green iff it appears infinitely many times on some run of $(P^{-\circ})^\infty$ with infinitely many broadcasts.
- light green iff it appears unboundedly many times between broadcasts on some run $\pi = \pi_0\pi_1\ldots$ of $(P^{-\circ})^\infty$ with infinitely many broadcasts, i.e., if for every $n \in \mathbb{N}_{>0}$ there are $i < j \in \mathbb{N}_{>0}$ such that $\pi_i \ldots \pi_j$ contains n transitions using this edge and no broadcast edges.
- dark green iff it is green but not light green.

Note that:

- light green edges are also green,
- dark green edges are exactly those green edges which satisfy that for every run of $(P^{-\circ})^\infty$ there is a bound on the number of times they appear between any two consecutive broadcasts,
- green edges only belong to components of $P^{-\circ}$ that are on the loop of the lasso,
- broadcast edges can only be dark green.

Example 6.2. Neither edge in $P^{-\circ}$ for the template P in Figure 1 is locally-reusable or green.

On the other hand, every edge in $P^{-\circ}$ for the template P in Figure 2 is dark green (and none are locally-reusable).

It turns out that determining the type of an edge is decidable; this is a non-trivial problem, and we dedicate Section 7 to solving it.

We now characterize the edge types in terms of witnessing cycles in $(P^{-\circ})^\infty$. Recall the definition of legal configuration and path (Definition 5.4).

Lemma 6.3. *An edge e of $P^{-\circ}$ is:*

i. *locally-reusable iff it appears on a legal cycle C_e of $(P^{-\circ})^\infty$ that has no broadcast.*
ii. *green iff it appears on a legal cycle C_e of $(P^{-\circ})^\infty$ that has broadcasts.*
iii. *light green iff it appears on a legal cycle D_e of $(P^{-\circ})^\infty$ that has no broadcast, that is contained in a legal cycle C_e with broadcasts;*
iv. *dark green iff it appears on a legal cycle C_e of $(P^{-\circ})^\infty$ that has broadcasts, but not on any cycle without broadcasts that is contained in a cycle with broadcasts.*

Proof. Observe that it is enough to prove the first three items.

For the 'if' directions, let $n \in \mathbb{N}_{>0}$ be the number of processes in C_e (i.e., C_e is a cycle in $(P^{-\circ})^n$), and recall that since C_e is legal, its source configuration f is in $(P_i^{-\circ})^n$ for some i. Hence, by Lemma 5.8 (Loading), a configuration g such that $g\big|_{[n]} = f$ (i.e., the first n processes of g form the configuration f) can be reached from an initial configuration of $(P^{-\circ})^\infty$; then, for items (i) and (ii), we can simply pump C_e forever (with the extra processes in g moving only on broadcasts). For item (iii), C_e is pumped in the following way: for every $i \in \mathbb{N}_{>0}$, at the i'th repetition of the outer cycle C_e we pump the inner cycle D_e for i times.

The 'only if' directions follow from the observation that every run in $(P^{-\circ})^\infty$ involves only finitely many processes, and thus only finitely many distinct configurations, all of which are legal (by Remark 5.5). □

Lemma 6.3 implies that every light green edge is locally-reusable, whereas a dark green edge may or may not be locally-reusable.[8] The following lemma states that we can assume that the cycles in Lemma 6.3 that witness broadcasts all have the same number of broadcasts.

Lemma 6.4. *There is a number K such that for every green, light green, or dark green edge e, the cycle C_e mentioned in items ii), iii) and iv) in Lemma 6.3 can be taken to contain exactly K broadcasts.*

Proof. Apply Lemma 6.3 to all the relevant edges in $P^{-\circ}$ and obtain cycles, say $C_{e_1}, C_{e_2}, \cdots, C_{e_l}$. Suppose C_{e_i} has k_i broadcasts. Let K be the least-common-multiple of the k_is. By repeating cycle C_{e_i} for K/k_i times, we obtain a witnessing cycle with exactly K broadcasts. □

We now informally describe the structure of the automaton $\mathcal{B}$. It is made of three copies of $\mathcal{A}$ (called $\mathcal{B}^{init}$, $\mathcal{B}^{grn}$, $\mathcal{B}^{loc}$) as follows: $\mathcal{B}^{init}$ is an exact copy of $\mathcal{A}$; the copy $\mathcal{B}^{grn}$ has only the green edges left; and $\mathcal{B}^{loc}$ has only the locally-reusable edges left (and in particular does not have broadcast edges). Furthermore, for every edge (s, σ, s') in $\mathcal{B}^{init}$ we add two new edges, both with the same source as the original edge, but one going to the copy of s' in $\mathcal{B}^{grn}$, and one to the copy of s' in $\mathcal{B}^{loc}$. The initial states of $\mathcal{B}$ are the initial states of $\mathcal{B}^{init}$. The (single) counter increments at every transition in $\mathcal{B}^{init}$, increment at every dark green rendezvous edge in $\mathcal{B}^{grn}$, and resets at every broadcast edge in $\mathcal{B}^{grn}$.

Here is the formal definition.

Definition 6.5 (B-Automaton $\mathcal{B}$)**.** Let us introduces the process template:

$$P = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S, I, R, \lambda \rangle$$

and let $P^{-\circ} = \langle AP, \Sigma_{\mathsf{rdz}} \cup \{\mathfrak{b}\}, S^{-\circ}, I^{-\circ}, R^{-\circ}, \lambda^{-\circ} \rangle$ be its unwinding. Define the B-automaton $\mathcal{B} = \langle \Sigma, S', I', R', G, cc \rangle$ as follows:

- $\Sigma = 2^{AP}$,
- $S' = \{init, grn, loc\} \times S^{-\circ}$,
- $G = S'$, i.e., the Büchi condition is always satisfied,
- $I' = \{init\} \times I^{-\circ}$,
- The transition relation R' is $\delta_{init} \cup \delta_{grn} \cup \delta_{loc}$ and the counter function cc are defined as follows. For every transition $e = (s, \sigma, t) \in R^{-\circ}$
  (1) δ_{init} contains the transitions $\tau = ((init, s), \lambda^{-\circ}(s), (i, t))$ for every $i \in \{init, grn, loc\}$; and $cc(\tau) = \mathsf{inc}$.
  (2) δ_{grn} contains the transition $\tau = ((grn, s), \lambda^{-\circ}(s), (grn, t))$ only if e is green; $cc(\tau) = \mathsf{inc}$ if e is a dark green rendezvous edge, $cc(\tau) = \mathsf{reset}$ if e is a broadcast edge, and otherwise $cc(\tau) = \mathsf{skip}$;
  (3) δ_{loc} contains the transition $\tau = ((loc, s), \lambda^{-\circ}(s), (loc, t))$ only if e is locally-reusable; $cc(\tau) = \mathsf{skip}$.

Since transitions of $\mathcal{B}$ are induced by transitions of $P^{-\circ}$, we call transitions of $\mathcal{B}$ broadcast, light green, etc., based on the classification of the corresponding transition of $P^{-\circ}$. Figure 7 depicts a high level view of the B-automaton constructed from a process template P, assuming that (s, σ_1, t) is a dark green rendezvous edge, (s, σ_2, u) is a green broadcast edge, (s, σ_3, v) is a light green rendezvous edge, and (s, σ_4, w) is a locally-reusable edge (for some $\sigma_1, \sigma_2, \sigma_3$, and σ_4).

[8] This overlap is the reason that we decided to use the term "locally-reusable" instead of naming these edges by another color.

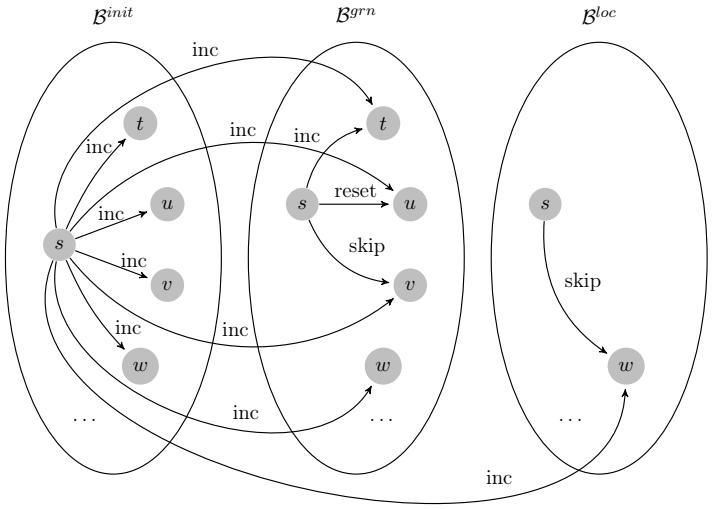


Figure 7: A high level view of the B-automaton construction.

The rest of this section is concerned with proving that the construction is correct:

Theorem 6.6. *For every RB-template P the language of B-automaton $\mathcal{B}$ is* EXEC-INF(P^∞).

Right-to-left direction. Take $\alpha \in$ EXEC-INF(P^∞). By Corollary 5.7 we have that EXEC(P^∞) = EXEC$((P^{-\circ})^\infty)$. Thus, there is some $\pi \in P^{-\circ}$ such that $\alpha = \lambda^{-\circ}(\pi)$. We now show that there is an accepting run of $\mathcal{B}$ on α. First note that by Definition 5.11 and Proposition 5.9, for every prefix of α there is a finite run on that prefix that remains in $\mathcal{B}^{init}$. There are two cases: either π contains infinitely many broadcast transitions or not. If it does not, then from some point on all edges on π are locally-reusable. Thus, at that point, the automaton can move from $\mathcal{B}^{init}$ to $\mathcal{B}^{loc}$. The resulting run is accepting since the counter is never incremented in $\mathcal{B}^{loc}$. On the other hand, if π has infinitely many broadcast transitions, then from some point on, all its edges are green. Thus, at that point, the automaton can move from $\mathcal{B}^{init}$ to $\mathcal{B}^{grn}$. Observe that the counter is reset on every broadcast edge and it is only incremented on dark green edges, which by Def. 6.1, appear only boundedly many times between broadcasts.

Outline of left-to-right direction. Let Ω be an accepting run in $\mathcal{B}$ on input α. By Corollary 5.7 it is enough to construct a run π in $(P^{-\circ})^\infty$ whose projection on process 1 has labeling α. Let β be the run in $P^{-\circ}$ induced by Ω (recall that every transition of the automaton is induced by a transition of $P^{-\circ}$). The construction of π is guided by having process 1 trace β. We decompose $\beta = \beta' \cdot \beta''$ where β' corresponds to the finite prefix of the run Ω that stays in $\mathcal{B}^{init}$.

In order to trace β', we use the techniques in Section 5 for finite traces. This leaves us with the task of tracing β'' which contains either only locally reusable edges or only green edges. First, observe that tracing a broadcast edge is easy since we can simply append a global broadcast transition to the run π being constructed. On the other hand, for each rendezvous edge e we will assign multiple groups of processes to help process 1 traverse e

(the number of groups is discussed later). Each group associated with e has the property that it can trace a *cycle* C_e in which edge e is taken at some point, say by process p_e. So, if e is the next edge that process 1 should take, we progress some group along the cycle C_e to the point where e should be taken, then process 1 swaps places with process p_e (this is virtual, and merely re-assigns process ids); and then the group takes the next transition along the cycle C_e, and so process 1 takes e. Note that in order for a group to be available to assist process 1 again in the future, it has to be 'reset', i.e., put back to the same position just before the edge e was taken. This is done differently, depending on the type of e. If e is locally-reusable, then so are all subsequent edges $f, g, \ldots$ that process 1 should take; so, since $C_e, C_f, C_g, \ldots$ do not contain any broadcast, the group can simply loop around C_e immediately after process 1 leaves C_e; when process 1 does leave C_e, it swaps with p_f in C_f, and so on. If e is light green, then it is on an inner cycle D_e, without broadcasts, of C_e (Lemma 6.3), so the group can loop around D_e after process 1 swaps out — we call this *recharging* — thus enabling it to help process 1 again even though it has not yet completed the outer cycle C_e. Finally, if e is dark green, this group will only be ready again after the whole cycle C_e is looped once more, which requires waiting for K broadcasts. Thus, until that happens, if process 1 needs to trace e it will need the help of another group associated with e. The key observation is that the number of these groups is bounded. The reason for this is that Ω is an accepting run, and thus one can deduce that there is a bound on the number of times a dark green edge is taken until the K broadcasts needed to complete the cycle C_e are taken.

Detailed proof of left-to-right direction. Let Ω be an accepting run in $\mathcal{B}$ on input α. Since every transition in $\mathcal{B}$ corresponds to a transition in $P^{-\circ}$, let β be the corresponding run in $P^{-\circ}$. Since Ω is an accepting run, it either gets trapped in $\mathcal{B}^{grn}$ or it gets trapped in $\mathcal{B}^{loc}$. Decompose $\Omega = \Omega' \cdot \Omega''$ accordingly, i.e., the prefix Ω' corresponds to the run until it first enters $\mathcal{B}^{loc}$ or $\mathcal{B}^{grn}$. Decompose $\alpha = \alpha' \cdot \alpha''$ and $\beta = \beta' \cdot \beta''$ accordingly.

We are required to construct a run of P^∞ whose projection on process 1 is labeled α. By Corollary 5.7 it is enough to construct a run π of $(P^{-\circ})^\infty$ whose projection onto process 1 is labeled α. We first construct a finite run ρ' of $(P^{-\circ})^\infty$ whose projection on process 1 is β'. Since Ω' stays inside $\mathcal{B}^{init}$ it is actually also an accepting run of $\mathcal{A}$ on α'. Thus, by Corollary 5.12, $\alpha' \in \text{EXEC-FIN}(P^\infty)$. By Corollary 5.7 it is also in $\text{EXEC-FIN}((P^{-\circ})^\infty)$, i.e., there is a run ρ' of $(P^{-\circ})^t$ for some number t of processes whose projection onto process 1 is α'. Let $P_l^{-\circ}$ be the component that the run ρ' ends in.

To complete the proof, we will construct an infinite path ρ'' of $(P^{-\circ})^n$ for some number n of processes, satisfying the following: 1) its projection on process 1 is β'' (note that this implies that ρ'' starts in a configuration where process 1 is in the same state as when it ended ρ'), and 2) it starts in a configuration in $P_l^{-\circ}$. To see why this is enough to complete the proof, proceed as follows in order to compose ρ' and ρ''. Apply Lemma 5.8 (Loading) to get a finite run ρ in $(P^{-\circ})^\infty$ that has the same number of broadcasts as ρ', and ends in a configuration that, when restricted to the first n processes, is the starting configuration of ρ''. Note that ρ' may use $m > n$ processes in order to achieve that. By Lemma 3.12, we can simultaneously simulate both ρ and ρ' in a run π' of $(P^{-\circ})^{t+m}$. Assume w.l.o.g. that the first t processes are simulating ρ', and that the next n processes are simulating the first n processes of ρ. Thus, at the final configuration of π', these n processes are exactly in the states needed to start simulating ρ'', and processes 1 and $t+1$ are in the same state. Thus, we can extend the simulation by letting process 1 exchange roles with process $t+1$ and

having processes $1, t+2, \ldots t+n$ simulate ρ'' (with all other processes doing nothing except responding to broadcasts). The resulting run π has the property that its projection onto process 1 is labeled α, as promised.

Constructing ρ''. First, assume w.l.o.g. that P (and thus also $P^{-\circ}$) has no self loops [9] — this is not essential, but simplifies some technicalities in the construction. Second, we differentiate between two cases, depending in which component Ω'' is trapped. We treat the case that Ω'' is trapped in $\mathcal{B}^{grn}$ (the case that Ω'' is trapped in $\mathcal{B}^{loc}$ is simpler, and does not use any new ideas). Note that we will ignore the technicality of keeping track of process numbers, as we find it distracts, rather then helps one understand the proof.

Let $E_{\mathsf{light\ green}}$ (resp. $E_{\mathsf{dark\ green}}$) be the set of light green (resp. dark green) edges that appear on β'', and note that these are the only edges that appear on it (by the fact that $\mathcal{B}^{grn}$ contains only green edges). For every edge $e \in E_{\mathsf{dark\ green}}$ (resp. $e \in E_{\mathsf{light\ green}}$), let C_e (resp. C_e, D_e) be the witnessing cycle(s) with exactly K broadcasts (for some fixed K) from Lemma 6.4, and assume w.l.o.g. that (a) every such cycle C_e starts in a configuration in the component $P_l^{-\circ}$ in which ρ' ends, and (b) that if e' is the first edge on β'', then e' appears in the first transition taken in the cycle $C_{e'}$. To see how to achieve (a) note that if e' and each subsequent edge is locally-reusable, then because each C_e does not contain any broadcast, each C_e is contained in $P_l^{-\circ}$. On the other hand, if the edges e are green, then e must be on the loop of the lasso, and so $P_l^{-\circ}$ is on the loop of the lasso (since e' is), and so since C_e contains at least one broadcast, it must go through $P_l^{-\circ}$. Since Ω is an accepting run, the counter is bounded on it. Thus, since in $\mathcal{B}^{grn}$ we increment the counter when reading a dark green edge, and reset it when reading a broadcast edge, we can pick $\mathfrak{m} \in \mathbb{N}_{>0}$ such that every dark green edge appears at most $\mathfrak{m}$ times on any section of β that contains K broadcasts.

Let the *designated occurrence* of e on C_e be defined as follows: if e is dark green then it is the first transition in C_e in which e is taken, and if e is light green then it is the first transition of C_e, that is also on the nested cycle D_e, in which e is taken. For every $h \in [K]$, let $C_e(h)$ denote the portion of C_e just after the $h-1$ broadcast up to (and including) the h broadcast. For h such that $C_e(h)$ contains the designated occurrence of e we divide $C_e(h)$ further into three pieces: $C_e(h,1)$ is the part up to the designated occurrence, $C_e(h,2)$ is the designated occurrence, and $C_e(h,3)$ is the remainder.

Take exactly enough processes to assign them to one copy G_e^1 of C_e for every $e \in E_{\mathsf{light\ green}}$, and $\mathfrak{m}$ copies $G_e^1, \ldots G_e^{\mathfrak{m}}$ of C_e for every $e \in E_{\mathsf{dark\ green}}$.

Given a group of processes G_e^i, for some i and e, we define the following operations:

- **flush**: G_e^i simulates (using Lemma 3.12) the portion of $C_e(h)$ which it has not yet simulated, up to but not including the broadcast;
- **load**: in case $C_e(h)$ contains the designated occurrence of e, then G_e^i simulates (using Lemma 3.12) the path $C_e(h,1)$;
- **swap**: we say that we *swap* process 1 into G_e^i to mean that process 1 and process j in G_e^i (where j is a process that takes the edge e in the designated occurrence of e on C_e) exchange their group associations. I.e., process 1 joins G_e^i, and process j takes its place in the former group of process 1;

[9] A template can be transformed, in linear time, to one without self loops (and the same set of executions) as follows: for every state s that has a self loop, add a new state $\hat{s}$ with the same labeling, replace every self loop (s, σ, s) with $(s, \sigma, \hat{s})$, and for every outgoing transition (s, σ', t), including self-loops, add the transition $(\hat{s}, \sigma', t)$.

- **recharge**: if e is light green, we say that we *recharge* C_e to mean that the group G_e^i simulates (using Lemma 3.12) tracing D_e until reaching (but not executing) the designated occurrence of e.
- **mark**: if e is dark green, we may mark a group G_e^i as *used* or *fresh* by setting or resetting a virtual flag.

Obviously (except for flush and mark), not every operation above can be taken at any time. In particular, a swap is allowed only at a time process 1 and j are in the same state, e.g., when process 1 is at the source of e and G_e^i has just been loaded (and thus process j is also at the source of e).

We are now ready to construct ρ''. The initial configuration of ρ'' is obtained by having all processes at the beginning of the cycles they were assigned to, and process 1 assigned to the group $G_{e'}^1$ where e' is the first edge on β'' (this can be done because e' appears in the first transition taken in the cycle $C_{e'}$). Note that this satisfies the requirement that ρ'' starts in a configuration in $P_l^{-\circ}$. The rest is done in blocks, where in the i'th block we extend ρ'' with a path ξ_i containing K broadcasts and whose projection on process 1 is the portion of β'' after the $(i-1)K$ broadcast up to and including the iK broadcast — which we call β_i''. The construction will ensure that ξ', defined as the concatenation of the ξ_is, weakly-simulates all the cycles of all the groups, and thus by Remark 3.14, will maintain the following invariant (†): at the configuration f at the start of each block, the processes in every group of the form G_e^i are in states corresponding to the initial configuration g of C_e (i.e., $f\big|_{G_e^i} = g$). The invariant obviously holds for the first block by our choice of the initial configuration of ρ''.

For $i \in \mathbb{N}_{>0}$, assume that blocks $< i$ have been constructed. We now describe how to build block i. First, **mark** all the groups as *fresh*, then proceed in K rounds by repeating the following algorithm for every $1 \le h \le K$. Let $e_1 \dots e_x$ be the prefix of β_i'' not yet traced by process 1, up to and including the next broadcast (obviously, the length x of this prefix depends on h). For every $j \in [x]$, if e_j is light green pick the group $G_{e_j}^1$; and if it is dark green pick the first *fresh* group from among the yet unpicked groups in $G_{e_j}^1 \dots G_{e_j}^{\mathfrak{m}}$, and **mark** this group as *used* (we can always pick a fresh group since — by our choice of $\mathfrak{m}$, and since β_i'' has exactly K broadcasts — there are at most $\mathfrak{m}$ occurrences of e_j in β_i''). Denote the group thus picked by $\mathbb{G}_{e_j}$, and let $\mathbb{G}_{e_0}$ denote whatever group process 1 is in at the beginning of the round.

(1) For $1 \le j \le x$ repeat:
   (a) **Load** the group $\mathbb{G}_{e_j}$ and **Swap** process 1 into it;
   (b) if $j < x$ then have group $\mathbb{G}_{e_j}$ simulate the transition $C_{e_j}(h, 2)$, with process 1 taking the edge e_j.
   (c) If $j > 1$, and e_{j-1} is a light green edge, then **recharge** $\mathbb{G}_{e_{j-1}}$.
(2) **Flush** all groups except $\mathbb{G}_{e_x}$ (that process 1 is currently in). Note that since e_x is a broadcast, the loading of $\mathbb{G}_{e_x}$ already put it in a flushed condition.
(3) Perform a broadcast (with process 1 taking the broadcast edge e_x).

We illustrate the execution of the above algorithm for one round in Figure 8. Here, we depict two dark green edges e_j and e_{j+1} of the run β'' along with the groups $\mathbb{G}_{e_j}$ and $\mathbb{G}_{e_j+1}$ that enable the simulation of these edges in the run ρ''. The illustration shows that each of the groups is first loaded (in order to prepare for the transition e_j resp. e_{j+1} to be taken), then process 1 is swapped into the group (allowing process 1 to execute the respective transition), after that transition e_j resp. e_{j+1} is taken, and finally each group is flushed to prepare for the next symmetric broadcast (in order to be ready to participate in the next

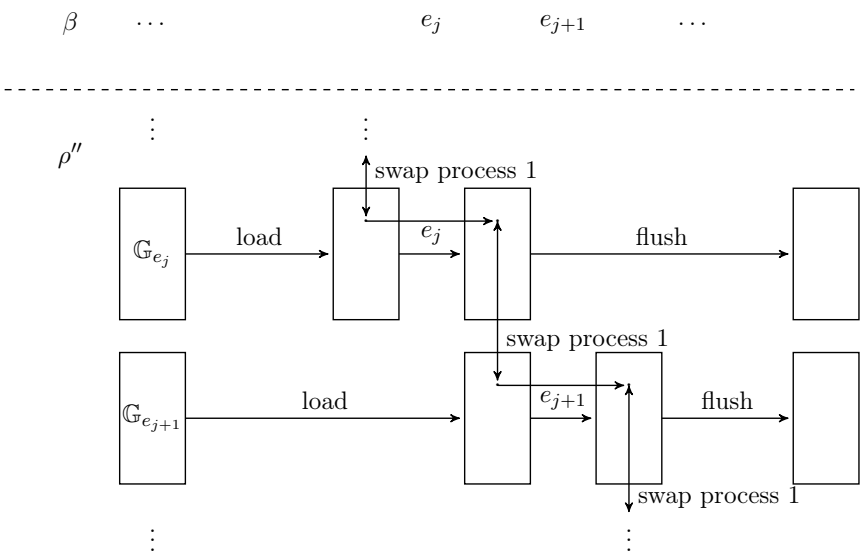


Figure 8: An illustration of how to simulate some dark green edges e_j and e_{j+1} of the run β'' in the run ρ'' using groups $\mathbb{G}_{e_j}$ and $\mathbb{G}_{e_j+1}$.

round of the simulation). Since the invariant (†) holds at the start of each block, it is easy to see that the algorithm can actually be executed. Indeed, the invariant ensures that the algorithm can **load** when needed (and thus, **swap** and **recharge** when needed). It is not hard to see that, as promised, the resulting path ξ_i weakly-simulates all the cycles of all the groups. To see that the projection of ξ_i on process 1 is β_i'', observe that the moves performed by process 1 in lines 1(b) and 3 of the algorithm trace exactly β_i''. Furthermore, process 1 is moved only in these lines since, by our assumption that $P^{-\circ}$ does not contain any self loop, every edge in β'' is different than the edge just before and just after it and thus, process 1 is never in a group when it is being loaded (except at the very beginning of the first block, in which case the **load** in line 1(a) of the algorithm does nothing since this group is already in a loaded position in the initial configuration of ρ'').

This completes the proof of Theorem 6.6.

In Section 7 we will show how to decide the type of the edges in $P^{-\circ}$ (Theorem 7.1) in polynomial time in the size of $P^{-\circ}$. Thus, we can build the B-automaton $\mathcal{B}$ in exponential time in the size of P. Combining this with Theorem 6.6 we get Theorem 3.9 that says that the PMCP for NBW/LTL specifications of RB-systems can be solved in EXPTIME.

Proof of Theorem 3.9. We reduce the PMCP problem to the emptiness problem for B-automata.

Given a process template P, and the corresponding B-automaton $\mathcal{B}$ (whose Büchi set is trivial), suppose the specification is given as an LTL formula φ (the case of NBW is given afterwards). Let $L(\mathcal{B})$ denote the language of $\mathcal{B}$, and let $L(\neg\varphi)$ denote the set of models of $\neg\varphi$. Then, every execution in EXEC-INF(P^∞) satisfies φ if and only if $L(\mathcal{B}) \cap L(\neg\varphi) = \emptyset$. Using Theorem 2.2, let $\mathcal{A}_{\neg\varphi}$ be an NBW accepting all models of $\neg\varphi$, and denote its states by Q and its Büchi set by G. Build the synchronous product of $\mathcal{A}_{\neg\varphi}$ and $\mathcal{B}$ to get an NBW with one counter, call it M, whose language is equal to $L(\mathcal{B}) \cap L(\mathcal{A}_{\neg\varphi})$. By Lemma 2.1, one can test whether $L(M)$ is empty in PTIME. Thus, this PMCP algorithm is exponential in

the size of the template P (since computing $\mathcal{B}$ can be done in time exponential in the size of P) and exponential in the size of φ (since computing $\mathcal{A}_{\neg\varphi}$ can be done in time exponential in the size of φ).

For the case that the specification is an NBW $\mathcal{A}$, we proceed in a similar way by noting that we can build an NBW $\mathcal{A}'$ for the complement of $L(\mathcal{A})$ in time exponential in the size of $\mathcal{A}$ (see for example [Saf88]). □

7. Deciding Edge Types

This section is dedicated to proving the following result:

Theorem 7.1. *Given a reachability-unwinding $P^{-\circ}$ of a process template P, the type (light green, dark green, and locally-reusable) of each edge e in $P^{-\circ}$ can be decided in PTIME (in the size of $P^{-\circ}$).*

We will develop the proof of Theorem 7.1 in several steps:

1. The starting point for the proof of Theorem 7.1 is the characterization of edge types in Lemma 6.3 through the existence (or lack thereof) of suitable cycles in $(P^{-\circ})^{\infty}$. In Subsection 7.1, we weaken this characterization using the notion of *pseudo-cycles*. Pseudo cycles are paths that start and end in configurations that are identical up to the renaming of processes, i.e., there are exactly the same number of processes in every state, though the identities of the processes in each of the states may differ. Pseudo-cycles can always be pumped to a cycle by iterating the pseudo-cycle until the initial configuration is reached again. Hence, pseudo-cycles can be seen as more compact representations of cycles. Importantly, we are able to obtain a bound on the number of broadcasts in pseudo-cycles, which we need for deciding edge types.

2. In order to be able to conveniently reason about pseudo-cycles we will work with counter abstractions of R-Systems: In Subsection 7.2, we define vector rendezvous systems (VRS) and their continuous relaxation, called continuous vector rendezvous systems (CVRS). The notions of VRSs and CVRS are inspired by the notion of Vector Addition Systems (VAS) [HP79], where configurations only store the number of processes for every process state but not the identity of the processes. VRSs are counter abstractions of R-Systems; in particular, pseudo-cycles of R-Systems correspond to cycles in VRSs and vice versa. CVRSs are a continuous relaxation of VRSs in which steps can be taken by a rational fraction. CVRSs have the advantage that we can characterize reachability and the existence of cycles in them by solving linear programming problems over the rationals. We will be able to work with CVRs instead of VRSs because we will show that we can scale a CVRSs cycle to a VRS cycle (as we are interested in the parameterized verification problem we can always scale the number of processes). We then reduce the existence of witnessing pseudo-cycles for the type of an edge to corresponding reachability statements for CVRSs, as outlined below.

3. In Subsection 7.3, we develop a characterization of reachability for CVRSs. This characterization will give rise to an equation system and a fixed point algorithm, which is the basis for our edge type computation procedure. The results in this subsection already allow us to compute the locally-reusable edges and the light green edges (under the assumption that the green edges are already known; the computation of the green edges is, however, only done in the next subsection).

4. In Subsection 7.4, we show how to decide whether an edge of $P^{-\circ}$ is green. This problem represents the main difficulty in deciding the type of an edge. We give an algorithm

that computes all green edge of $P^{-\circ}$ and prove its correctness. We further remark that, though not actually needed, one can derive a pseudo-cycle that contains all green edges from our procedure.

7.1. **Pseudo-cycles.**

Definition 7.2. Let P be an RB-template with state set S, and let $n \in \mathbb{N}_{>0}$. Two configurations f, f' of P^n are called *twins* if every state is covered by the same number of processes in f and f', i.e. $|f^{-1}(q)| = |f'^{-1}(q)|$ for every $q \in S$.

Let $g \circ h$ denote the composition $g(h(\cdot))$. Observe that f, f' are twins if and only if there is a (not necessarily unique) permutation $\theta : [n] \to [n]$ such that $f' = f \circ \theta$. Intuitively, θ maps each process in f' to a matching process in f (i.e., one in the same state). Thus, given a transition $f \xrightarrow{\sigma} g$, say t, in P^n, we denote by $t[\theta]$ the transition $f \circ \theta \xrightarrow{\sigma'} g \circ \theta$ resulting from replacing process i with process $\theta(i)$ in t; and having $\sigma' = \mathfrak{b}$ if $\sigma = \mathfrak{b}$, and $\sigma' = ((\theta^{-1}(i_1), a_1), \ldots, (\theta^{-1}(i_k), a_k))$ if $\sigma = ((i_1, a_1), \ldots, (i_k, a_k))$. Note that the rendezvous or broadcast action of the transition taken in t and $t[\theta]$ are the same — it is only the identities of the processes involved that are different. Extend θ to paths point-wise, i.e., if $\pi = t_1 t_2 \ldots$ is a path then define $\pi[\theta] = t_1[\theta] t_2[\theta] \ldots$.

Definition 7.3. A finite path π of an RB-system P^n is a *pseudo-cycle* if $\mathsf{src}(\pi)$ and $\mathsf{dst}(\pi)$ are twins.

Obviously, every cycle is also a pseudo-cycle, but not vice-versa. For example, for P in Figure 9, the following path in P^4 is a pseudo-cycle that is not a cycle: $(p,q,q,r) \xrightarrow{((3,c_1),(4,c_2))} (p,q,r,p) \xrightarrow{((2,c_1),(3,c_2))} (p,r,p,p) \xrightarrow{((3,a_1),(4,a_2))} (p,r,q,q)$.

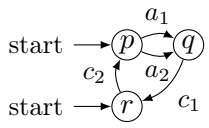


Figure 9: R-template.

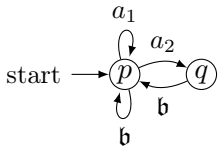


Figure 10: RB-template.

Remark 7.4. Similar to cycles, for which one can chose any point on the cycle as its start (and end) point, one can chose any point along a pseudo-cycle as the start point. Indeed, if C is a pseudo-cycle that starts in a configuration f and ends in a twin f', then given any configuration g along C we can obtain a new pseudo-cycle, that uses exactly the same edges (but with possibly different processes taking these edges) as follows: start in g and traverse the suffix of C until f', reassign process id's according to the permutation transforming f to f' and traverse the prefix of C from f to g using these reassigned processes to reach a twin g' of g.

The following immediate lemma states that every pseudo-cycle π can be pumped to a cycle.

Lemma 7.5. *Given a pseudo-cycle (resp. legal pseudo-cycle) π in P^n, and a permutation θ such that $\mathsf{dst}(\pi) = \mathsf{src}(\pi) \circ \theta$, then there is a $z \geq 1$ such that*

$$\pi[\theta^0]\pi[\theta^1]\pi[\theta^2]\dots\pi[\theta^{z-1}],$$

is a cycle (resp. legal cycle) in P^n, where θ^j denotes the composition of θ with itself j times.

Proof. We can choose z such that the permutation θ^z is the identity (recall that the set of permutations of a finite set forms a finite group, and that the order of every element in a finite group is finite, i.e., there is some $z \in \mathbb{N}_{>0}$ such that θ^z is the identity permutation). We observe that $\mathsf{dst}(\pi[\theta^0]\pi[\theta^1]\pi[\theta^2]\dots\pi[\theta^{z-1}]) = \mathsf{src}(\pi) \circ \theta^z = \mathsf{src}(\pi)$, and thus it is a cycle. □

Recall that n, r denote the prefix-length and period of $P^{\multimap}$, respectively. The following lemma states that we can assume that if an edge of $P^{\multimap}$ appears on a pseudo-cycle with broadcasts then it also appears on one with exactly r broadcasts. Knowing this bound will be crucial for decidability of edge types (in contrast, Lemma 6.3 only says that a bound exists), as well as for obtaining good complexity for deciding PMCP.

Lemma 7.6 (Spiral)**.** *An edge e appears on a legal pseudo-cycle D in $(P^{\multimap})^\infty$, which contains broadcasts, iff it appears on a legal pseudo cycle C of $(P^{\multimap})^\infty$, containing exactly r broadcasts and starting in a configuration in $P_n^{\multimap}$.*

Proof. Assume D is a legal pseudo-cycle in $(P^{\multimap})^m$, for some $m \in \mathbb{N}_{>0}$. By Remark 7.4 we can assume w.l.o.g. that D starts and ends in a configuration in $P_n^{\multimap}$. Observe that (by the lasso structure of $P^{\multimap}$) D must have lr broadcast transitions for some $l \in \mathbb{N}_{>0}$, and that after every r broadcasts all processes are in $P_n^{\multimap}$. Let f_0 be the initial configuration on D. For every $i \in [l]$, let f_i be the configuration in D just after ir broadcasts, and let ρ_i be the portion of D from f_{i-1} to f_i. Observe that ρ_i contains exactly r broadcasts. By Lemma 3.12, we can compose the paths $\rho_1, \dots \rho_l$ into a single path C in the system $(P^{\multimap})^{ml}$ with ml processes. It is not hard to see that C is a legal pseudo-cycle. Indeed, after r broadcasts the processes that were simulating ρ_i are in states that match the configuration $f_{(i+1) \bmod l}$, i.e., the configuration that the processes that simulate $\rho_{(i+1) \bmod l}$ started in. Obviously, e appears on C, which completes the proof. □

We now use the notion of a pseudo-cycle to give a characterization of the edges types that is easier to detect than the one given in Section 6:

Lemma 7.7. *An edge e of $P^{\multimap}$ is:*

i. *locally-reusable iff it appears on a legal pseudo-cycle C_e of $(P^{\multimap})^\infty$ without broadcasts.*
ii. *green iff it appears on a legal pseudo-cycle C_e of $(P^{\multimap})^\infty$ with r broadcasts.*
iii. *light green iff it appears on a legal pseudo-cycle C_e of $(P^{\multimap})^\infty$ without broadcasts that only uses green edges;*
iv. *dark green iff it is green and does not appear on a legal pseudo-cycle C_e of $(P^{\multimap})^\infty$ without broadcasts that only uses green edges.*

Proof. Item iv. follows from item iii. since green edges are partitioned to light green and dark green edges.

Items i. and ii. follow immediately from Lemmas 6.3 and 7.5 and the fact that every cycle is also a pseudo-cycle (for the 'only if' direction of item ii. use Lemma 7.6 to obtain a

pseudo-cycle with exactly r broadcasts). The same fact, combined with Lemma 6.3, gives the 'only if' direction of item iii..

For the 'if' direction of item iii., we claim that it is enough to show that: (†) there is a configuration f of $(P^{-\circ})^\infty$ and two pseudo-cycles C, D starting in f, such that D contains broadcasts, and C contains the edge e and does not contain any broadcast edge. Indeed, by Lemma 7.5, one can pump C, D to obtain cycles C', D' starting and ending in f, and the 'figure-eight' cycle obtained by concatenating C' to D' is (according to Lemma 6.3) a cycle witnessing that the edge e is $\mathsf{light\ green}$.

We now show that if e appears on a pseudo-cycle C_e satisfying item iii. then (†) holds. Let g be the initial configuration of C_e and let $H := \{s \mid g(s) > 0\}$ be the set of states of $P^{-\circ}$ for which there is some process in that state in g. Assume w.l.o.g. that there is no process that does not move on C_e (such processes can simply be removed), and for every $s \in H$, let e^s be some edge in $P^{-\circ}$ appearing in C_e whose source is s. Observe that by item iii. the edge e^s is green, and apply Lemma 6.3 to obtain a witnessing cycle C_{e^s} starting with some process in s. By Lemma 6.4, we can assume that the cycles C_{e^s} thus obtained for all states in H have the same number of broadcasts. Hence, by Lemma 3.12, we can run together $g(s)$ copies of C_{e^s}, for all s in H, in one cycle D. Let f be the starting configuration of D. Note that $f(s) \geq g(s)$ for every state s in $P^{-\circ}$, and obtain a pseudo-cycle C starting in f which simulates C_e (since C_e does not have any broadcast, processes that f has in excess of g can simply not move). The proof is complete by noting that f, C and D satisfy †. □

7.2. **Vector Rendezvous Systems.** We now formally introduce VRSs and CVRSs. We recall that k denotes the number of processes participating in a rendezvous, Σ_{actn} denotes a finite set of *rendezvous actions*, and $\Sigma_{\mathsf{rdz}} = \cup_{a \in \Sigma_{\mathsf{actn}}} \{a_1, \ldots, a_k\}$ denotes the *rendezvous alphabet*.

Given a finite set S, we can think of $\mathbb{Q}^S$ as the set of rational vectors of dimension $|S|$, and we use the elements of S as indices into these vectors. We also use the standard operations of vector addition and scalar multiplication. Finally, we compare vectors point-wise, i.e., given $c, c' \in \mathbb{Q}^S$ we say that $c \leq c'$ iff $c(s) \leq c'(s)$ for all $s \in S$.

Continuous Vector Rendezvous System (CVRS). A *continuous vector rendezvous system* (CVRS) is a tuple $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R \rangle$, where S is a finite set of states and $R \subseteq S \times \Sigma_{\mathsf{rdz}} \times S$ is a finite set of *transitions*. The *configurations* of $\mathcal{V}$ are the vectors $\mathbb{Q}^S_{\geq 0}$. For a transition $t = (p, \sigma, q)$, we denote by $\mathsf{rdz}(t) = \sigma$ its rendezvous symbol, by $\mathsf{src}(t) = p$ the *source* state of t, and by $\mathsf{dst}(t) = q$ the *destination* state of t. Also, we denote by $\mathsf{out}(t) \in \{0,1\}^S$ the vector that has a 1 entry at index p and zero entries otherwise, and by $\mathsf{in}(t) \in \{0,1\}^S$ the vector that has a 1 entry at index q and zero entries otherwise. We now define what it means for a CVRS to take a step.

Definition 7.8 (Step of a CVRS)**.** Given a k-tuple $\mathsf{t} = (t_1, \ldots, t_k)$ of transitions in R, the CVRS $\mathcal{V}$ can *step* with *multiplicity* $\alpha \in \mathbb{Q}_{>0}$ from a configuration c to a configuration c' using transitions $\mathsf{t} = (t_1, \ldots, t_k)$, denoted $c \xrightarrow{t_1,\ldots,t_k:\alpha} c'$, or $c \xrightarrow{\mathsf{t}:\alpha} c'$, if:

(i) there is an action $a \in \Sigma_{\mathsf{actn}}$ such that $\mathsf{rdz}(t_i) = a_i$ for all $i \in [k]$,
(ii) $c \geq \alpha \sum_{i=1}^k \mathsf{out}(t_i)$, and
(iii) $c' = c + \alpha \sum_{i=1}^k (\mathsf{in}(t_i) - \mathsf{out}(t_i))$.

A step is said to *synchronize* on a. We say that a transition $t \in R$ *participates* in a step $c \xrightarrow{t_1,\ldots,t_k:\alpha} c'$ if $t = t_i$ for some $i \in [k]$.

A *trace* of $\mathcal{V}$ is a sequence of steps $c_1 \xrightarrow{\mathrm{t}_1:\alpha_1} c_2 \xrightarrow{\mathrm{t}_2:\alpha_2} \cdots c_n$. We say that a configuration c' is *reachable* from configuration c, denoted $c \to^\star c'$, if there is a trace $c_1 \xrightarrow{\mathrm{t}_1:\alpha_1} c_2 \xrightarrow{\mathrm{t}_2:\alpha_2} \cdots c_n$, with $c = c_1$ and $c' = c_n$.

Vector Rendezvous System (VRS). A *vector rendezvous system (VRS)*, is a restriction of a CVRS, where the set of configurations is $\mathbb{N}^S$ and all steps are restricted to have multiplicity $\alpha = 1$. Given any VRS $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R\rangle$, the *relaxation* of $\mathcal{V}$ is the same tuple interpreted as a CVRS. Since a natural number is also rational, it is obvious that any VRS $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R\rangle$ is also a CVRS.

Note that every trace of a VRS is also a trace of its relaxation, and that the relaxation has more traces than the VRS since every rational multiple of a step in the VRS is a step in its relaxation.

Remark 7.9. Every R-template $P = \langle AP, \Sigma_{\mathsf{rdz}}, S, I, R, \lambda\rangle$ defines a VRS $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R\rangle$ with the same set of rendezvous alphabet, states, and transitions. These two systems are closely related. Intuitively, $\mathcal{V}$ is an abstraction of P^∞ in the sense that it does not keep track of the state of every individual process, but only keeps track of the number of processes in every state. More formally, every configuration $f \in P^\infty$ induces a configuration c of $\mathcal{V}$, called its *counter representation*, defined as $c(s) := |f^{-1}|(s)$ for $s \in S$, i.e., $c(s)$ is the number of processes of f that are in state s. Furthermore, $c \xrightarrow{t_1,\cdots,t_k} c'$ is a step of $\mathcal{V}$ if and only if there is a global transition (f, σ, f') in P^∞ such that c and c' are the counter representations of f and f' respectively, and $t_1, \cdots, t_k$ are exactly the rendezvous edges taken by the k active processes in the transition (f, σ, f').

We now define operations for manipulating traces. Given a trace $\xi := c_1 \xrightarrow{\mathrm{t}_1:\alpha_1} c_2 \xrightarrow{\mathrm{t}_2:\alpha_2} \cdots c_n$ of a CVRS $\mathcal{V}$, we define the following two operations:

(1) Multiplication by a scalar $0 < \gamma$: Let $\gamma \otimes \xi$ be the trace $\gamma c_1 \xrightarrow{\mathrm{t}_1:\gamma\alpha_1} \gamma c_2 \xrightarrow{\mathrm{t}_2:\gamma\alpha_2} \cdots \gamma c_n$.
(2) Addition of a constant configuration c: let $c \oplus \xi$ be the trace $c + c_1 \xrightarrow{\mathrm{t}_1:\alpha_1} c + c_2 \xrightarrow{\mathrm{t}_2:\alpha_2} \cdots c + c_n$.

It is not hard to see, by consulting the definition of a step, that $\gamma \otimes \xi$ and $c \oplus \xi$, are indeed traces of $\mathcal{V}$. Note, however, that multiplying by a scalar $\gamma < 0$ would not yield a trace, and that adding a vector c that is not a configuration (i.e., which has negative coordinates) may sometimes also not yield a trace — either because intermediate points may not be configurations (due to having some negative coordinates), or since condition (ii) in the definition of a step (Definition 7.8) is violated. Finally, traces in CVRSs have a *convexity property* that states that by taking a fraction $0 < \gamma < 1$ of each step of a trace $c \to^\star c'$ one obtains a trace from c to the convex combination $(1-\gamma)c + \gamma c'$:

Proposition 7.10 (convexity)**.** *Let $\xi := c_1 \xrightarrow{\mathrm{t}_1:\alpha_1} c_2 \xrightarrow{\mathrm{t}_2:\alpha_2} \cdots c_n$ be a trace of a CVRS $\mathcal{V}$, and let $0 < \gamma < 1$ be rational. Define configurations $c'_i := \gamma c_i + (1-\gamma)c_1$ for $1 < i \le n$. Then $\xi' := c_1 \xrightarrow{\mathrm{t}_1:\gamma\alpha_1} c'_2 \xrightarrow{\mathrm{t}_2:\gamma\alpha_2} \cdots c'_n$ is a trace of $\mathcal{V}$.*

Proof. Simply observe that ξ' is the trace $((1-\gamma)c_1) \oplus (\gamma \otimes \xi)$. □

In the following two lemmas we combine Remark 7.9 with Lemma 7.7 in order to rephrase the characterization of edge types in terms of the existence of certain CVRSs traces. We begin by characterizing the locally-reusable edges and the light green edges (relative to green edges):

Lemma 7.11. *An edge e of $P^{-\circ}$ is locally-reusable iff e participates in a step of a cyclic trace of some component $P_i^{-\circ}$ considered as CVRS, i.e., iff e participates in a step of a trace $\xi := c \to^\star c$ of CVRS $P_i^{-\circ}$. Moreover, e is* light green *iff ξ uses only* green *edges.*

Proof. We start with the 'only if' direction. By Lemma 7.7 there is a legal pseudo-cycle C, which contains e, and which does not contain broadcasts. Because C does not contain broadcasts, we have that all configurations of C are in some component $P_i^{-\circ}$ of $P^{-\circ}$. Let f be the initial configuration of C, and let c be its counter representation. By Remark 7.9, C induces a trace $\xi := c \to^\star c$ in the VRS corresponding to $P_i^{-\circ}$, and e participates in a step of ξ. As every VRS is a CVRS the claim follows.

For the other direction, let e participate in a step of some trace $\xi := c \to^\star c$ of the CVRS corresponding to $P_i^{-\circ}$. Let x be the least common multiple of all the denominators that appear in the multiplicities of any step, or any coordinate of any configuration of ξ. Consider the trace $x \otimes \xi := xc \to^\star xc$. Observe that all configurations on this trace are in $\mathbb{N}^{S_i}$, and that all steps on it are taken with an integer multiplicity. By replacing every step that uses a multiplicity $y \in \mathbb{N}_{>0}$ with y consecutive steps each with multiplicity 1, we obtain a trace $\varrho := xc_i \to^\star xc_i'$ in $P_i^{-\circ}$ considered as VRS. By Remark 7.9, there is a corresponding pseudo-cycle C of $P_i^{-\circ}$, and e appears on C. The claim then follows from Lemma 7.7. □

We now turn to characterizing the green edges.

For the statement of the lemma, recall (see Section 5) that the template $P^{-\circ}$ is built from component templates arranged in a lasso structure. Let $\mathcal{I} := \{n, n+1, \ldots, n+r-1 = m\}$ be the set of indices of the components on the noose of $P^{-\circ}$, and for every $i \in \mathcal{I}$ define $next(i)$ (resp. $prev(i)$) to be the component number immediately following i (resp. preceding i) along the noose. Note that, in particular, $next(n+r-1) = n$ and $prev(n) = n+r-1$.

Lemma 7.12. *An edge e of $P^{-\circ}$ is* green *iff, for every $i \in \mathcal{I}$: (i) there is a subset $\mathcal{T}_i$ of the transitions of $P_i^{-\circ}$, and a subset B_i of the broadcast transitions from $P_i^{-\circ}$ to $P_{next(i)}^{-\circ}$; (ii) there are coefficients $\alpha_t \in \mathbb{Q}_{>0}$ for every $t \in B_i$; and (iii) there is a trace $\xi_i := c_i \to^\star c_i'$ of the CVRS of $P_i^{-\circ}$ using exactly the transitions $\mathcal{T}_i$; such that: $e \in \cup_{i\in\mathcal{I}}(B_i \cup \mathcal{T}_i)$, and for all $q \in S_i^{-\circ}$ the following holds:*

(1) $c_i(q) = \sum_{t \in B_{prev(i)}, \mathsf{dst}(t)=q} \alpha_t$.
(2) $c_i'(q) = \sum_{t \in B_i, \mathsf{src}(t)=q} \alpha_t$.

Proof. By Lemma 7.7, it is enough to show that e appears on a pseudo-cycle C, with r broadcasts, of $(P^{-\circ})^\infty$ iff the conditions of the lemma hold. First, assume that such a pseudo-cycle C exists. For every $i \in I$, let f_i (resp. f_i') be the configurations of C in $P_i^{-\circ}$ just after (resp. just before) a broadcast, let C_i be the sub-path of C from f_i to f_i', let $\mathcal{T}_i$ be the process transitions appearing on C_i. By Remark 7.9, C_i induces a trace $\xi_i := c_i \to^\star c_i'$ of the CVRS of $P_i^{-\circ}$, using exactly the transitions in $\mathcal{T}_i$, where c_i, c_i' are the counter representations of f_i, f_i'. Consider now the global broadcast transition of C from f_i' to $f_{next(i)}$, let B_i be the set of local broadcast transitions that participate in it, and for every $t \in B_i$ let α_t to be the number of processes in f_i' that take t. It is easy to see that the conditions of the lemma are satisfied.

For the other direction, assume that sets $\mathcal{T}_i$ and B_i, coefficients α_t, and traces ξ_i satisfying the conditions of the lemma exist. Using the same argument as in the proof of Lemma 7.11 — by taking x to be the least common multiple of all the denominators that appear in the multiplicities of any step or any coordinate of any configuration of these traces, as well as any of the coefficients α_t — we can obtain from each trace ξ_i a trace $\varrho_i := xc_i \to^\star xc_i'$

in the VRS corresponding to $P_i^{-\circ}$. We can now build the required pseudo-cycle C in r rounds. We start by (arbitrarily) picking some $i_i \in \mathcal{I}$, and a configuration f_{i_1} whose counter representations is xc_{i_1}. At each round j, we extend C by concatenating a path (obtained from ϱ_{i_j} by Remark 7.9) from f_{i_j} to a configuration f'_{i_j} whose counter representation is xc'_{i_j}; we then append a global broadcast transition in which each edge $t \in B_{i_j}$ is taken by exactly $x\alpha_t$ processes, resulting in a configuration $f_{i_{j+1}}$ whose counter representation is $xc_{next(i_j)}$ (this is possible by equations (1) and (2) in the conditions of the lemma).

It follows that, at the end of the last round, C is a path in $(P^{-\circ})^\infty$ from f_{i_1} to $f_{i_{r+1}}$, whose counter representations are xc_{i_1} and $xc_{next^r(i_1)} = xc_{i_1}$, respectively. Thus, C is a pseudo-cycle. □

In Lemmas 7.11 and 7.12 we have related edge-types to the existence of certain CVRS traces. Developing a criterion for the existence of CVRS traces is the subject of the next subsection. Having developed this characterization, we can replace the existence of CVRS traces with this criterion in the two lemmas mentioned. This will then allow us to present our algorithms for deciding edge-types using linear programming.

7.3. **Characterization of Reachability in CVRSs.** Our aim in this section is to develop a characterization of the existence of a trace between two configurations c, c' in a CVRS which can be used as a basis for an efficient algorithm. Looking at Definition 7.8 of a step, one can see that the task is very simple if c' is reachable from c in one step. However, in the general case, a trace from c to c' may have a very large number of steps, so we have to find a way to avoid reasoning about each step individually. The following proposition is our starting point for summarizing a large number of steps by equations, and it follows immediately from the definitions of a step and a trace:

Proposition 7.13. *Let $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R\rangle$ be a CVRS, and let $\xi := c \to^\star c'$ be a trace in it. For every $t \in R$, let $\alpha_t \in \mathbb{Q}_{\geq 0}$ be the sum of the multiplicities of all the steps in ξ in which t participates. Then:*

(1) $c' = c + \sum_{t\in R} \alpha_t \left(\mathsf{in}(t) - \mathsf{out}(t)\right)$;
(2) *For every rendezvous action $a \in \Sigma_{\mathsf{actn}}$, and every $i, j \in [k]$, we have that*

$$\sum_{t\in R, \mathsf{rdz}(t)=a_i} \alpha_t = \sum_{t\in R, \mathsf{rdz}(t)=a_j} \alpha_t$$

We will see that the inverse of Proposition 7.13 also holds, provided one adds some suitable conditions, as follows. The equations of Proposition 7.13 ensure that the 'accounting' in a trace is done correctly, i.e., that the transitions can be allocated to steps — item (2), corresponding to requirement (i) in Definition 7.8 — and that added to the vector c they yield the vector c' — item (1), corresponding to requirement (iii) in that definition. In order to extend Proposition 7.13 to a full characterization of reachability, we need to address the last element in the definition of a step (requirement (ii)) which states, for every state s, that $c(s)$ is at least the multiplicity of the step times the number of transitions that participate in the step whose source is s. Observe that if $c(s) > 0$ this condition can always be satisfied if α is small enough; however, if $c(s) = 0$, and there is some $i \in [k]$ such that $\mathsf{src}(t_i) = s$, then the condition is necessarily violated. The characterization we now develop will thus make a distinction between those states for which $c(s) > 0$ and those for which $c(s) = 0$.

We will use the following terminology: For a vector $c \in \mathbb{Q}_{\geq 0}^S$, let the *support* of c be the set $(c)^{\neq 0} := \{s \in S \mid c(s) > 0\}$. We will also make use of the following notation: for a set of transitions R, we lift src and dst as follows: $\mathsf{src}(R) = \{\mathsf{src}(t) \mid t \in R\}$ and $\mathsf{dst}(R) = \{\mathsf{dst}(t) \mid t \in R\}$.

The following lemma states that we can summarize a large number of steps between two configurations in case the support of the two configurations is the same and no step increases the support:

Lemma 7.14. *Let $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R\rangle$ be a CVRS. Let $c, c' \in \mathbb{Q}_{\geq 0}^S$ be configurations and let $\alpha_t \in \mathbb{Q}_{\geq 0}$ be coefficients for every $t \in R$ such that:*

(1) $c' = c + \sum_{t \in R} \alpha_t \left(\mathsf{in}(t) - \mathsf{out}(t)\right)$;

(2) *For every rendezvous action $a \in \Sigma_{\mathsf{actn}}$, and every $i, j \in [k]$, we have that*

$$\sum_{t \in R, \mathsf{rdz}(t) = a_i} \alpha_t = \sum_{t \in R, \mathsf{rdz}(t) = a_j} \alpha_t$$

(3) $(c)^{\neq 0} = (c')^{\neq 0}$;

(4) *For $R' := \{t \in R \mid \alpha_t > 0\}$ we have $\mathsf{src}(R') \subseteq (c')^{\neq 0}$ and $\mathsf{dst}(R') \subseteq (c)^{\neq 0}$.*

Then, we have $c \to^\star c'$ and all transitions in R' participate in some step of this trace.

Proof. Equations (1) and (2) give us a good starting point for showing the existence of a trace from c to c'. Indeed, (2) allows us to group the transitions in R' into steps such that each transition $t \in R'$ is taken with a combined multiplicity of α_t; and (1) guarantees that concatenating all these steps will take us from c to c'. It remains to find a way to make sure each of these steps satisfies requirement (ii) in the definition of a step (Definition 7.8). We begin, however, by ignoring this problem. That is, we will consider *quasi-steps* and *quasi-traces*. Formally, quasi-steps are like steps except that they are between arbitrary vectors in $\mathbb{Q}^S$ (as opposed to steps which were defined only for vectors in $\mathbb{Q}_{\geq 0}^S$), and that they do not have to satisfy requirement (ii) of Definition 7.8. We will denote quasi-steps using $\Longrightarrow$. A quasi-trace is simply a sequence of quasi-steps where the destination of each quasi-step in the sequence is the source of the next one. We complete the proof by first constructing a quasi-trace ϱ from c to c', and then showing how to turn ϱ into a trace.

We construct ϱ using the following algorithm. If $c = c'$ then we are done. Otherwise, at round 0, set ϱ to the empty quasi-trace, and $v_1 := c$ be the source of the first quasi-step to be constructed. At round ≥ 1, do the following:

(1) pick a transition t whose α_t is minimal among all edges in R'; let a_i be $\mathsf{rdz}(t)$;

(2) let $t_i := t$, and for every $j \in [k] \setminus \{i\}$ pick some $t_j \in R'$ with $\mathsf{rdz}(t_j) = a_j$;

(3) extend ϱ with the quasi-step $v_i \xRightarrow{t_1, \ldots, t_k : \alpha_t} v_{i+1}$;

(4) for every $j \in [k]$, subtract α_t from α_{t_j}, and if the resulting α_{t_j} is zero then remove t_j from R'. If $R' = \emptyset$ stop and output ϱ.

Since at least one transition (namely the transition t picked in step 1) is removed at the end of each round, the algorithm stops after at most $|R'|$ rounds. It is also easy to see that item (2) of the lemma is an invariant of the algorithm (i.e., it holds using the updated values at the end of each round). This, together with our choice of t, ensures that we are always able to find the required transitions in the second step of every round. Finally, observe that the resulting final quasi-trace ϱ uses every transition t in (the initial) R' with a combined multiplicity which is exactly α_t and thus, by item (1) of the lemma, we have that ϱ ends in c' as promised. It remains to show how to derive from ϱ a trace from c to c'.

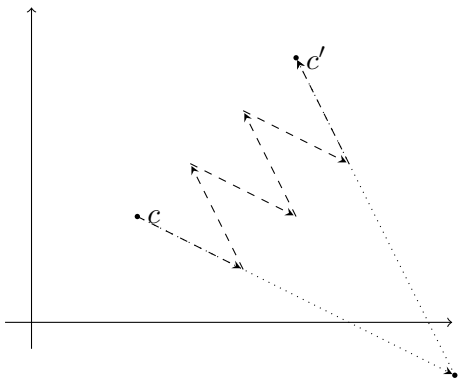


Figure 11: A graphical representation in two dimensions of the replacement of quasi-steps (in dotted arrows) by steps (in dashed arrows). In this example $h = 3$ and $m = 2$.

Let m be the number of quasi-steps in ϱ. For $j \in [m]$, denote the transitions taken in the j-th quasi-step of ϱ by $t_1^j, \ldots, t_k^j$, denote the multiplicity used by α^j, and the destination reached by v_j (also set $v_0 := c$). Consider $c = v_0, v_1, \ldots, v_m = c'$ as points in the $|S|$-dimensional Euclidean space. Let $H := (c)^{\neq 0} = (c')^{\neq 0}$ (by item (3) of the lemma). We note that every point on the line segment L between c and c' also has support H (being a convex combination of c and c'). It follows that L does not touch any of the hyperplanes corresponding to the states in H (i.e., the hyperplanes defined by the equations $v(s) = 0$, for $s \in H$). Let $x > 0$ be the minimum of the Euclidean distances between L and any of these hyperplanes[10], let y be the maximum over $j \in m$ of the multiplicities α^j, and let $h \in \mathbb{N}_{>0}$ be large enough to satisfy $\frac{myk}{h} \leq x$.

Construct a new quasi-trace ϱ' from c to c' by starting at c, and repeating h times the following: for $j = 1, \ldots, m$, extend ϱ' by taking a quasi-step using the transitions $t_1^j, \ldots, t_k^j$ with multiplicity α^j/h. This construction is illustrated in Figure 11. We claim that ϱ' is actually a trace, i.e., that for every $s \in S$, and every point v_{lm+j} on ϱ' (where $0 \leq l < h$, and $0 \leq j < m$) we have that condition (ii) in Definition 7.8 is satisfied, namely, that $v_{lm+j}(s) \geq o_{lm+j}(s)$, where o_{lm+j} is the vector $\frac{\alpha^j}{h} \sum_{i=1}^{k} \mathsf{out}(t_i^j)$. Observe that this not only guarantees that each quasi-step is a step, but also that all points on ϱ' are configurations (i.e., have no negative coordinates). Consider first the case of $s \in S \setminus H$. By item (4) of the lemma, $\mathsf{src}(R') \subseteq H$ and $\mathsf{dst}(R') \subseteq H$, and thus $v_{lm+j}(s) = c(s) = 0 = o_{lm+j}(s)$. For $s \in H$, note that $o_{lm+j}(s) \leq \frac{yk}{h}$, and thus it is enough to show that $v_{lm+j}(s) - \frac{yk}{h} \geq 0$. Observe that, for every $0 \leq l < h$, the point reached after taking lm quasi-steps of ϱ' is $c + \frac{l}{h}(c' - c)$, which is a point on L. Hence, by our choice of x, we have that $v_{lm}(s) \geq x$. Also note that each quasi-step on ϱ' can decrease $v_{lm}(s)$ by at most $\frac{yk}{h}$, thus, for every $0 \leq j < m$, we have that $v_{lm+j}(s) - \frac{yk}{h} \geq v_{lm}(s) - \frac{myk}{h} \geq x - x = 0$. □

The above lemma already allows us to characterize the locally-reusable edges, as well as the light green edges (relative to green edges):

[10]Recall that in a Euclidean space the distance between a hyperplane and a line-segment, that does not touch it, is positive.

Lemma 7.15. *Given a component $P_i^{\multimap}$ of $P^{\multimap}$, an edge e of $P_i^{\multimap}$ is locally-reusable iff there are coefficients $\alpha_t \in \mathbb{Q}_{\geq 0}$ for every $t \in R_i^{\multimap}$, such that:*

(1) $0 = \sum_{t \in R_i^{\multimap}} \alpha_t \, (\mathit{in}(t) - \mathit{out}(t))$;
(2) *For every rendezvous action $a \in \Sigma_{\mathsf{actn}}$, and every $j, h \in [k]$, we have that* $\sum_{t \in R_i^{\multimap}, \mathsf{rdz}(t) = a_j} \alpha_t = \sum_{t \in R_i^{\multimap}, \mathsf{rdz}(t) = a_h} \alpha_t$;
(3) $\alpha_e > 0$.

Moreover, e is light green iff we further require that the set $\{t \in R_i \mid \alpha_t > 0\}$ only contains green edges.

Proof. For the 'only if' direction, we observe that by Lemma 7.11 there is a trace $\xi := c \to^\star c$ of the CVRS $P_i^{\multimap}$ such that e participates in a step of ξ. Moreover, ξ uses only green edges in case e is light green. By Proposition 7.13, we can derive coefficients $\alpha_t \in \mathbb{Q}_{\geq 0}$ that satisfy the conditions of the lemma.

For the 'if' direction, let $R' = \{t \in R_i \mid \alpha_t > 0\}$ and $H = \mathsf{src}(R') \cup \mathsf{dst}(R')$. Set $c \in \mathbb{Q}_{\geq 0}^{S_i}$ to the configuration defined by $c(q) = 1$, if $q \in H$, and $c(q) = 0$, otherwise. Then, apply Lemma 7.14 and obtain a trace $\xi := c \to^\star c$ of the CVRS $P_i^{\multimap}$, such that e participates in a step of ξ. By Lemma 7.11, the type (locally-reusable, or light green) of e follows. □

Lemma 7.15 gives rise to a PTIME algorithm for computing the locally-reusable and light green edges (in case we already know the green edges):

Corollary 7.16. *Let e be an edge of $P^{\multimap}$. We can decide in PTIME (in the size of $P^{\multimap}$), whether e is locally-reusable, and whether e is light green (assuming we already know which edges are green).*

Proof. For every edge e of $P^{\multimap}$, we need to find a solution that satisfies the equations of Lemma 7.15. Such a solution can be found by linear programming over the rationals, which is in PTIME. □

The rest of this section will be dedicated to addressing the problem of deciding whether an edge is green or not. Observe that — unlike locally-reusable and light green edges for which a witnessing pseudo-cycle is characterized by Lemma 7.11 using a cyclic trace in a single CVRS — the characterization of the witnessing pseudo-cycle for a green edge in terms of CVRS traces (as given by Lemma 7.12) involves non-cyclic traces in multiple different CVRSs. Hence, our first step is to develop a general characterization of reachability in CVRS which extends the characterization given in Lemma 7.14 to the case of traces whose source and destination configurations may have a different support. We begin with the following property of the support:

Lemma 7.17. *Let $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R \rangle$ be a CVRS and let $c, c_1, c_2 \in \mathbb{Q}_{\geq 0}^{S}$ be configurations of it. Then:*

(1) *If $c \to^\star c_1$ and $c \to^\star c_2$ then there is a trace $c \to^\star c_3$ with $(c_3)^{\neq 0} = (c_1)^{\neq 0} \cup (c_2)^{\neq 0}$;*
(2) *If $c_1 \to^\star c$ and $c_2 \to^\star c$ then there is a trace $c_3 \to^\star c$ with $(c_3)^{\neq 0} = (c_1)^{\neq 0} \cup (c_2)^{\neq 0}$.*

Proof. Let $c_3 = \frac{1}{2}c_1 + \frac{1}{2}c_2$, and observe that $(c_3)^{\neq 0} = (c_1)^{\neq 0} \cup (c_2)^{\neq 0}$. It remains to show the desired traces between c and c_3. For the first item, let $\xi_1 := c \to^\star c_1$, and $\xi_2 := c \to^\star c_2$. A trace from c to c_3 is obtained by concatenating $\frac{1}{2}c \oplus (\frac{1}{2} \otimes \xi_1)$ and $\frac{1}{2}c_1 \oplus (\frac{1}{2} \otimes \xi_2)$. For the second item, let $\xi_1 := c_1 \to^\star c$, and $\xi_2 := c_2 \to^\star c$. A trace from c_3 to c is obtained by concatenating $\frac{1}{2}c_1 \oplus (\frac{1}{2} \otimes \xi_2)$ and $\frac{1}{2}c \oplus (\frac{1}{2} \otimes \xi_1)$. □

Let $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R\rangle$ be a CVRS, let $R' \subseteq R$, and let $c \in \mathbb{Q}_{\geq 0}^S$ be a configuration. We say that a set $H \subseteq S$ is *forward (resp. backward) R'-accessible from c* if there is a trace $c \to^\star c'$ (resp. $c' \to^\star c$), with all of its steps using only transitions from R', such that $(c')^{\neq 0} = H$. We define $\mathsf{forw}(c, R')$ (resp. $\mathsf{back}(c, R')$) to be the union of all sets $H \subseteq S$ that are forward (resp. backward) R'-accessible from c.

Remark 7.18. Observe that Lemma 7.17 implies that $\mathsf{forw}(c, R')$ is forward R'-accessible from c, and that $\mathsf{back}(c, R')$ is backward R'-accessible from c. It follows that $\mathsf{forw}(c, R')$ (resp. $\mathsf{back}(c, R')$) is the maximal subset of S that is forward (resp. backward) R'-accessible from c.

Proposition 7.19. *Given a CVRS $\mathcal{V} = \langle \Sigma_{rdz}, S, R\rangle$, a subset $R' \subseteq R$, and a configuration c, the sets $\mathsf{forw}(c, R')$ and $\mathsf{back}(c, R')$ can be computed in PTIME.*

Proof. We present a fixed point algorithm for computing $\mathsf{forw}(c, R')$; computing $\mathsf{back}(c, R')$ is done in a symmetric fashion. Construct an increasing chain $H_0 \subsetneq H_1 \subsetneq \cdots$ of sets $H_i \subseteq S$, until a larger set cannot be found, as follows. Let $H_0 := (c)^{\neq 0}$. For each i, we check if there is an action $a \in \Sigma_{\mathsf{actn}}$, and there are transitions $t_1, \ldots, t_k \in R'$ with $\mathsf{rdz}(t_j) = a_j$ for all $j \in [k]$, such that $\mathsf{src}(\{t_1, \ldots, t_k\}) \subseteq H_i$ and $\mathsf{dst}(\{t_1, \ldots, t_k\}) \not\subseteq H_i$. In case there are such transitions, we set $H_{i+1} := H_i \cup \mathsf{dst}(\{t_1, \ldots, t_k\})$; otherwise, we are done and have $H_i = \mathsf{forw}(c, R')$.

The correctness of the algorithm is demonstrated as follows. First, to see that for every round i, the set H_i is contained in $\mathsf{forw}(c, R')$, proceed by induction on i. Note that, by the induction hypothesis, there is a trace $\xi := c \to^\star c_i$ with $(c_i)^{\neq 0} = H_i$, and that this trace can be extended to a trace $\xi' := c \to^\star c_i \xrightarrow{t_1,\ldots,t_k:\alpha} c_{i+1}$, with $(c_{i+1})^{\neq 0} = \mathsf{dst}(\{t_1, \ldots, t_k\}) \cup H_i = H_{i+1}$, by choosing $0 < \alpha < \frac{1}{k} \cdot \min_{q \in (c_i)^{\neq 0}}\{c_i(q)\}$. To see that the algorithm outputs $\mathsf{forw}(c, R')$, and not a proper subset of it, let $\xi := c \to^\star c'$ be a trace such that $(c')^{\neq 0} = \mathsf{forw}(c, R')$. Assume by way of contradiction that the support of some configuration along ξ is not contained in the output of the algorithm, and let c_{i+1} be the first such configuration on ξ. Consider the step $c_i \xrightarrow{t_1,\cdots,t_k:\alpha_i} c_{i+1}$ and note that $\mathsf{src}(\{t_1, \ldots, t_k\}) \subseteq (c_i)^{\neq 0}$. Let $P_0, P_1, P_2, \cdots, P_n$ be the sequence of sets computed by the algorithm. By minimality of i, there is a j such that $(c_i)^{\neq 0} \subseteq P_j \subseteq P_n$. Thus, $\mathsf{src}(\{t_1, \ldots, t_k\}) \subseteq P_n$ and $\mathsf{dst}(\{t_1, \ldots, t_k\}) \not\subseteq P_n$. But this contradicts the termination condition of the algorithm. □

The following simple property of the operators forw and back will be useful:

Proposition 7.20. *Let $\mathcal{V} = \langle \Sigma_{rdz}, S, R\rangle$ be a CVRS and let $\xi := c \to^\star c'$ be a trace of $\mathcal{V}$. Let R' be the set of transitions that participate in steps of ξ, and let Configs be the set of configurations that appear in ξ. Then, $\mathsf{forw}(c, R') = \mathsf{back}(c', R') = \bigcup_{c^\circ \in Configs}(c^\circ)^{\neq 0} = (c)^{\neq 0} \cup \mathsf{dst}(R') = (c')^{\neq 0} \cup \mathsf{src}(R')$.*

Proof. For every configuration c° that appears on ξ, the prefix $c \to^\star c^\circ$ of ξ, and the suffix $c^\circ \to^\star c'$ of ξ, obviously only use transitions from R'. Hence, $\bigcup_{c^\circ \in Configs}(c^\circ)^{\neq 0} \subseteq \mathsf{forw}(c, R')$ and $\bigcup_{c^\circ \in Configs}(c^\circ)^{\neq 0} \subseteq \mathsf{back}(c', R')$; as well as $\bigcup_{c^\circ \in Configs}(c^\circ)^{\neq 0} \subseteq (c)^{\neq 0} \cup \mathsf{dst}(R')$ and $\bigcup_{c^\circ \in Configs}(c^\circ)^{\neq 0} \subseteq (c')^{\neq 0} \cup \mathsf{src}(R')$. For the other direction, observe that $\mathsf{forw}(c, R') \subseteq (c)^{\neq 0} \cup \mathsf{dst}(R') \subseteq \bigcup_{c^\circ \in Configs}(c^\circ)^{\neq 0}$ and, similarly, $\mathsf{back}(c', R') \subseteq (c')^{\neq 0} \cup \mathsf{src}(R') \subseteq \bigcup_{c^\circ \in Configs}(c^\circ)^{\neq 0}$. □

Remark 7.21. It is worth noting that for every configuration c and set of transitions R', we have that $\mathsf{forw}(c, R') \subseteq (c)^{\neq 0} \cup \mathsf{dst}(R')$ (resp. $\mathsf{back}(c, R') \subseteq (c)^{\neq 0} \cup \mathsf{src}(R')$); however, only in case there is a path from c (resp. to c), that uses exactly the transitions in R', do the reverse inclusions also hold.

We are now ready to state a full characterization of reachability in CVRSs:

Theorem 7.22. *Let $\mathcal{V} = \langle \Sigma_{\mathsf{rdz}}, S, R\rangle$ be a CVRS. A configuration $c' \in \mathbb{Q}_{\geq 0}^S$ is reachable from a configuration $c \in \mathbb{Q}_{\geq 0}^S$ iff there are coefficients $\alpha_t \in \mathbb{Q}_{\geq 0}$ for every $t \in R$ such that:*

(1) $c' = c + \sum_{t\in R} \alpha_t \left(\mathit{in}(t) - \mathit{out}(t)\right)$;
(2) *For every rendezvous action $a \in \Sigma_{\mathsf{actn}}$, and every $i, j \in [k]$, we have that $\sum_{t\in R,\mathsf{rdz}(t)=a_i} \alpha_t = \sum_{t\in R,\mathsf{rdz}(t)=a_j} \alpha_t$;*
(3) *For $R' := \{t \in R \mid \alpha_t > 0\}$ we have that $\mathsf{src}(R') \subseteq \mathsf{back}(c', R')$, $\mathsf{dst}(R') \subseteq \mathsf{forw}(c, R')$, and $\mathsf{forw}(c, R') = \mathsf{back}(c', R')$.*

Proof. For the forward direction, take a trace $\xi := c \to^\star c'$. For every $t \in R$, let α_t be the sum of the multiplicities of the steps of ξ in which t participates. By Proposition 7.13, we have that condition (1) and (2) of the lemma are satisfied. Condition (3) holds by applying Proposition 7.20 to ξ.

For the reverse direction, assume that there are coefficients α_t such that conditions (1)-(3) are satisfied. Let $H := \mathsf{forw}(c, R') = \mathsf{back}(c', R')$. By Remark 7.18, we can obtain traces $\xi_1 := c \to^\star c_1$ and $\xi_2 := c_2 \to^\star c'$, using only transitions from R', such that $(c_1)^{\neq 0} = H = (c_2)^{\neq 0}$. It remains to show that c_2 is reachable from c_1. For every transition $t \in R$, let α_t^1 (resp. α_t^2) be the sum of the multiplicities of the steps of ξ_1 (resp. ξ_2) in which t participates, and let $\gamma_t = \alpha_t - \alpha_t^1 - \alpha_t^2$. By the convexity property (Proposition 7.10) we can assume w.l.o.g. that: (§) α_t^1 and α_t^2 are small enough such that $\gamma_t > 0$ for all $t \in R'$ (simply apply the convexity property to ξ_1 and ξ_2 with a small enough γ to obtain, if needed, replacement ξ_1, ξ_2, c_1, c_2). By Proposition 7.13, we get that $c_1 = c + \sum_{t\in R'} \alpha_t^1 \left(\mathsf{in}(t) - \mathsf{out}(t)\right)$, and $c' = c_2 + \sum_{t\in R'} \alpha_t^2 \left(\mathsf{in}(t) - \mathsf{out}(t)\right)$. Combining the last two equalities with condition (1) of the Lemma, and rearranging terms, we get:

$$c_2 = c_1 + \sum_{t\in R'} \gamma_t \left(\mathsf{in}(t) - \mathsf{out}(t)\right) \qquad (\dagger)$$

Again, by Proposition 7.13, we have for every $l \in \{1, 2\}$:

$$\forall a \in \Sigma_{\mathsf{actn}}, i, j \in [k] : \sum_{t\in R',\mathsf{rdz}(t)=a_i} \alpha_t^l = \sum_{t\in R',\mathsf{rdz}(t)=a_j} \alpha_t^l,$$

Together with condition (2) we thus get that:

$$\forall a \in \Sigma_{\mathsf{actn}}, i, j \in [k] : \sum_{t\in R',\mathsf{rdz}(t)=a_i} \gamma_t = \sum_{t\in R',\mathsf{rdz}(t)=a_j} \gamma_t \qquad (\ddagger)$$

By (†) and (‡), and our choice of c_1, c_2, the requirements of Lemma 7.14 are satisfied for c_1, c_2 and coefficients γ_t. Hence, c_2 is reachable from c_1. □

Remark 7.23. The characterization in Theorem 7.22 is an adaptation of the characterization of reachability in continuous vector addition systems from [FH15]; this characterization leads to a PTIME decision procedure for reachability. We have redeveloped this characterization result here for CVRSs for the following reason: One can convert VRSs resp. CVRSs into VASs resp. CVASs, resulting, however, in a potentially exponential number of transitions. That is because for every k-wise rendezvous action a, we have to create a transition of the VAS resp. CVAS, for every possible combination of transitions labelled by $a_1, \ldots, a_k$. The resulting number of transitions is exponential in k. By redeveloping the results for CVRSs,

however, we directly obtain (basically the same) characterization result for CVRSs, which leads to a PTIME decision procedure for reachability (which is used as part of Algorithm 1).

7.4. **An Algorithm for Finding the green Edges.** In this section we present a PTIME algorithm for deciding the existence of pseudo-cycles with r broadcasts in $(P^{\multimap})^{\infty}$. The algorithm will be developed based on the following characterization which is an immediate consequence of Lemma 7.12, using Theorem 7.22 to characterize the traces ξ_i of this lemma. Recall that $\mathcal{I}$ is the set of indices of the components on the noose of $P^{\multimap}$.

Corollary 7.24. *An edge e of $P^{\multimap}$ is* green *iff, for every $i \in \mathcal{I}$: (i) there is a subset $\mathcal{T}_i$ of the transitions of $P_i^{\multimap}$, and a subset B_i of the broadcast transitions from $P_i^{\multimap}$ to $P_{next(i)}^{\multimap}$, with $e \in \cup_{i\in\mathcal{I}}(B_i \cup \mathcal{T}_i)$; (ii) there are coefficients $\alpha_t \in \mathbb{Q}_{>0}$ for every $t \in \mathcal{T}_i \cup B_i$; such that:*

(1) $c_i' = c_i + \sum_{t\in\mathcal{T}_i} \alpha_t\,(\mathsf{in}(t) - \mathsf{out}(t))$, *where c_i, c_i' are defined, for every $q \in S_i^{\multimap}$, by:* $c_i(q) := \sum_{t\in B_{prev(i)},\mathsf{dst}(t)=q} \alpha_t$, *and* $c_i'(q) := \sum_{t\in B_i,\mathsf{src}(t)=q} \alpha_t$;
(2) *for every rendezvous action $a \in \Sigma_{\mathsf{actn}}$, and every $j, h \in [k]$, we have that* $\sum_{t\in\mathcal{T}_i,\mathsf{rdz}(t)=a_j} \alpha_t = \sum_{t\in\mathcal{T}_i,\mathsf{rdz}(t)=a_h} \alpha_t$;
(3) $\mathsf{dst}(\mathcal{T}_i) \subseteq \mathsf{forw}(c_i, \mathcal{T}_i)$, $\mathsf{src}(\mathcal{T}_i) \subseteq \mathsf{back}(c_i', \mathcal{T}_i)$, *and* $\mathsf{forw}(c_i, \mathcal{T}_i) = \mathsf{back}(c_i', \mathcal{T}_i)$.

Corollary 7.24 gives rise to Algorithm 1 for computing the set of green edges of $P^{\multimap}$.

We remark that the algorithm presented here for computing green edges differs from our original algorithm in [ARZS15]. Here, we use an algorithm that is inspired by, and extends, Algorithm 2 in [FH15] for deciding the reachability of a target configuration from an initial configuration in continuous vector addition systems. The extension is in two ways: first from vector addition systems to CVRSs, and second by adding machinery for handling broadcasts.

Theorem 7.25. *Deciding if an edge of $P^{\multimap}$ is* green *can be done in* PTIME.

Proof. We begin by making a couple of important observations. In each iteration of the main loop of the algorithm we need to find a solution to a constraint system forming a linear programming problem over the rationals, with no objective function, whose canonical form is: $Ax = 0, x \geq 0$ (for some matrix A and vector x of variables), such that the solution is maximal with respect to the number of non-zero variables α_t. Given that linear programs of this canonical form have the property that the sum of any two solutions is itself a solution, we have that: (†) one can find a maximal solution x by looking, for every $t \in \mathcal{T}_i \cup B_i$, for a solution to the system $Ax = 0, x \geq 0, \alpha_t > 0$, and adding together all the solutions that were found; (‡) for every variable α_t of the system $Ax = 0, x \geq 0$, if there is a solution in which $\alpha_t > 0$, then $\alpha_t > 0$ in every maximal solution.

We now show that Algorithm 1 outputs exactly the set of green edges of $P^{\multimap}$.

To see that every edge output by the algorithm is green, we will apply Corollary 7.24 (direction 'if') to the sets $\mathcal{T}_i$ and B_i and the coefficients α_t from the solution obtained in the last iteration of the algorithm. Observe that at this last iteration all the sets have reached a fixed point. Hence, in particular, for every $t \in \mathcal{T}_i \cup B_i$ we have that $\alpha_t > 0$, and (§) $\mathsf{dst}(\mathcal{T}_i) \subseteq H_i$, $\mathsf{src}(\mathcal{T}_i) \subseteq H_i$. Since the constraint system in the algorithm exactly matches requirements (1) and (2) in Corollary 7.24, the only thing we have to show before we can apply this corollary is that also condition (3) holds. Observe that, by §, it is enough to show that at the last iteration $\mathsf{forw}(c_i, \mathcal{T}_i) = H_i$ and $\mathsf{back}(c_i', \mathcal{T}_i) = H_i$. We will show the first equality, the second is shown in a symmetric way. Consider the following chain of inequalities: $H_i \subseteq \mathsf{forw}(c_i, \mathcal{T}_i) \subseteq (c_i)^{\neq 0} \cup \mathsf{dst}(\mathcal{T}_i) \subseteq (c_i)^{\neq 0} \cup H_i \subseteq H_i$. The first containment is by the

Input: $P^{\multimap}$.

Initialize — for every $i \in \mathcal{I}$ do:
$\mathcal{T}_i :=$ all rendezvous transitions of $P_i^{\multimap}$;
$B_i :=$ all broadcast transitions from $P_i^{\multimap}$ to $P_{next(i)}^{\multimap}$;

Repeat: For every $i \in \mathcal{I}$: take variables $c_i(q), c_i'(q)$ for every $q \in S_i^{\multimap}$, and α_t for every $t \in \mathcal{T}_i \cup B_i$.
Find a solution to the following constraint system, such that the number of non-zero variables α_t is maximal:

- $\alpha_t \geq 0$ for every $t \in \mathcal{T}_i \cup B_i$;
- $c_i(q) = \sum_{t \in B_{prev(i)}, \mathsf{dst}(t)=q} \alpha_t$, for every $q \in S_i^{\multimap}$;
- $c_i'(q) = \sum_{t \in B_i, \mathsf{src}(t)=q} \alpha_t$, for every $q \in S_i^{\multimap}$;
- $c_i' = c_i + \sum_{t \in \mathcal{T}_i} \alpha_t \, (\mathsf{in}(t) - \mathsf{out}(t))$;
- $\sum_{t \in \mathcal{T}_i, \mathsf{rdz}(t)=a_j} \alpha_t = \sum_{t \in \mathcal{T}_i, \mathsf{rdz}(t)=a_h} \alpha_t$ for every $a \in \Sigma_{\mathsf{actn}}$ and $j, h \in [k]$.

Let $H_i := \mathsf{forw}(c_i, \mathcal{T}_i) \cap \mathsf{back}(c_i', \mathcal{T}_i)$;
Let $\mathcal{T}_i := \{t \in \mathcal{T}_i \mid \alpha_t > 0\} \cap \{t \in \mathcal{T}_i \mid \mathsf{src}(t) \in H_i \wedge \mathsf{dst}(t) \in H_i\}$;
Let $B_i := \{t \in B_i \mid \alpha_t > 0\}$;
Until neither $\mathcal{T}_i$ nor B_i change, for any $i \in \mathcal{I}$.

Output: $\bigcup_{i \in \mathcal{I}} B_i \cup \mathcal{T}_i$

Algorithm 1: Algorithm for computing all edges of $P^{\multimap}$ that can appear in a pseudo-cycle with broadcasts of $(P^{\multimap})^{\infty}$.

definition of H_i, the second by Remark 7.21, the third by §, and the last by the following argument: for $q \in (c_i)^{\neq 0}$, if $q \in \mathsf{src}(\mathcal{T}_i) \cup \mathsf{dst}(\mathcal{T}_i)$ then $q \in H_i$ by §; otherwise, $q \in (c_i')^{\neq 0}$ (by the constraint $c_i' = c_i + \sum_{t \in \mathcal{T}_i} \alpha_t \, (\mathsf{in}(t) - \mathsf{out}(t))$), and since by definition $(c_i)^{\neq 0} \subseteq \mathsf{forw}(c_i, \mathcal{T}_i)$ and $(c_i')^{\neq 0} \subseteq \mathsf{back}(c_i', \mathcal{T}_i)$, we have that $q \in \mathsf{forw}(c_i, \mathcal{T}_i) \cap \mathsf{back}(c_i', \mathcal{T}_i) = H_i$.

To see that every green edge e of $P^{\multimap}$ is output by the algorithm, apply Corollary 7.24 (direction 'only if') to e to obtain sets $\tilde{\mathcal{T}}_i$ and $\tilde{B}_i$, and coefficients $\tilde{\alpha}_t > 0$, satisfying the conditions of the corollary. We claim that if $\tilde{\mathcal{T}}_i \cup \tilde{B}_i \subseteq \mathcal{T}_i \cup B_i$ holds, for every $i \in \mathcal{I}$, at the beginning of an iteration (which is the case at initialization), then it also does at its end. Note that this would conclude the proof since $e \in \cup_{i \in \mathcal{I}} (\tilde{B}_i \cup \tilde{\mathcal{T}}_i)$ by condition (i) of the corollary. To prove the claim, note that if $\tilde{\mathcal{T}}_i \cup \tilde{B}_i \subseteq \mathcal{T}_i \cup B_i$, for every $i \in \mathcal{I}$, then the coefficients $\tilde{\alpha}_t$ induce a (not necessarily maximal) solution to the constraint system. Hence, by observation (‡) at the beginning of the proof, every maximal solution α_t, c, c' of this system will have $\alpha_t > 0$ for every $t \in \tilde{\mathcal{T}}_i \cup \tilde{B}_i$. The claim follows by showing that $\mathsf{src}(\tilde{\mathcal{T}}_i) \cup \mathsf{dst}(\tilde{\mathcal{T}}_i) \subseteq H_i$. To see this, observe that $\alpha_t > 0$ for every $t \in \tilde{B}_i$ implies (by the definition of c_i and c_i' in condition (1) of the corollary, and the corresponding constraint in the algorithm) that $(\tilde{c}_i)^{\neq 0} \subseteq (c_i)^{\neq 0}$ and $(\tilde{c}_i')^{\neq 0} \subseteq (c_i')^{\neq 0}$. Hence, $\mathsf{forw}(\tilde{c}_i, \tilde{\mathcal{T}}_i) \subseteq \mathsf{forw}(c_i, \mathcal{T}_i)$ and $\mathsf{back}(\tilde{c}_i', \tilde{\mathcal{T}}_i) \subseteq \mathsf{back}(c_i', \mathcal{T}_i)$. Since, by condition (3) of the corollary, $\mathsf{src}(\tilde{\mathcal{T}}_i) \cup \mathsf{dst}(\tilde{\mathcal{T}}_i) \subseteq \mathsf{forw}(\tilde{c}_i, \tilde{\mathcal{T}}_i) = \mathsf{back}(\tilde{c}_i', \tilde{\mathcal{T}}_i)$ we are done.

It remains to show that the algorithm runs in polynomial time. Since in every round at least one transition is removed from either $\mathcal{T}_i$ or B_i, for some i, the main loop repeats linearly many times in the size of $P^{\multimap}$. By observation (†) at the beginning of the proof, finding a

maximal solution to the constraint system of each iteration can be done by solving a linear number of linear programming problems (of polynomial size) over the rationals, which is in PTIME. Since, by Proposition 7.19, calculating $\mathsf{forw}(c_i, \mathcal{T}_i)$ and $\mathsf{back}(c'_i, \mathcal{T}_i)$ is also in PTIME, we conclude that the whole algorithm runs in polynomial time. □

Theorem 7.25, together with Corollary 7.16, yield the promised proof to the main theorem (Theorem 7.1) of this section.

8. Conclusion

We have established the decidability and complexity of the PMCP for safety and liveness properties of RB-systems, which are polynomially inter-reducible with discrete-timed networks. The lower and upper complexity bounds for safety properties are tight. We leave open the problem of whether our EXPTIME upper-bound for liveness properties is tight. We note that the PSPACE lower bound for safety properties also implies a PSPACE lower bound for liveness properties. The EXPTIME upper bound is established by (repeatedly) solving an exponentially sized linear program. As linear programming is known to be PTIME-complete, it seems unlikely that our techniques can be improved to show a PSPACE upper bound.

A further direction for future research concerns whether our results for the discrete-time model can be lifted to the continuous-time model without a distinguished controller (note that PMCP for continuous time networks with a distinguished controller is undecidable [AJ03]). For example, time-bounded invariance and time-bounded response properties (expressed as MTL formulae) hold on the discrete-time model iff they hold with the continuous-time model [HMP92], whereas [OW03] establish several results on to the use of digitization techniques for timed automata.

9. Acknowledgments

This work is partially supported by the Austrian Science Fund (FWF): P 32021, the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF), the PRIN project RIPER (No.20203FFYLK), the project MOST (CUP D93C22000400001) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, the FWF project AUTOSARD: "Automated Sublinear Amortised Resource Analysis of Data Structures" No. P36623 and the project VASSAL: "Verification and Analysis for Safety and Security of Applications in Life" funded by the European Union under Horizon Europe WIDERA Coordination and Support Action/Grant Agreement No. 101160022.

References

[AAC+18] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Radu Ciobanu, Richard Mayr, and Patrick Totzke. Universal safety for timed Petri nets is PSPACE-complete. In *Proceedings of 29th International Conference on Concurrency Theory, CONCUR*, volume 118 of *LIPIcs*, pages 6:1–6:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.6.

[ADM04] Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, LICS*, pages 345–354, July 2004. doi:10.1109/LICS.2004.1319629.

[ADR+16] Parosh Aziz Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. Parameterized verification of time-sensitive models of ad hoc network protocols. *Theoretical Computer Science*, 612:1–22, 2016. doi:10.1016/j.tcs.2015.07.048.

[AEJK23] Étienne André, Paul Eichler, Swen Jacobs, and Shyam Lal Karra. Parameterized verification of disjunctive timed networks. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 124–146. Springer, 2023. doi:10.1007/978-3-031-50524-9_6.

[AJ03] Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003. doi:10.1016/S0304-3975(01)00330-9.

[AJKR14] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *Verification, Model Checking, and Abstract Interpretation*, pages 262–281. Springer, 2014. doi:10.1007/978-3-642-54013-4_15.

[AKR+18] Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. *Distributed Computing*, 31(3):187–222, 2018. doi:10.1007/s00446-017-0302-6.

[Alu99] Rajeev Alur. Timed automata. In *Computer Aided Verification*, pages 8–22. Springer, 1999. doi:10.1007/3-540-48683-6_3.

[AMRZ22] Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. Verification of agent navigation in partially-known environments. *Artificial Intelligence*, 308:103724, 2022. doi:10.1016/j.artint.2022.103724.

[AR16] Benjamin Aminof and Sasha Rubin. Model checking parameterised multi-token systems via the composition method. In *Proceedings of Automated Reasoning - 8th International Joint Conference, IJCAR 2016*, volume 9706 of *Lecture Notes in Computer Science*, pages 499–515. Springer, 2016. doi:10.1007/978-3-319-40229-1_34.

[ARZ15] Benjamin Aminof, Sasha Rubin, and Florian Zuleger. On the expressive power of communication primitives in parameterised systems. In *Proceedings of Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20*, volume 9450 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 2015. doi:10.1007/978-3-662-48899-7_22.

[ARZS15] Benjamin Aminof, Sasha Rubin, Florian Zuleger, and Francesco Spegni. Liveness of parameterized timed networks. In *Proceedings of Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015*, volume 9135 of *Lecture Notes in Computer Science*, pages 375–387. Springer, 2015. doi:10.1007/978-3-662-47666-6_30.

[AST18] Parosh Aziz Abdulla, A. Prasad Sistla, and Muralidhar Talupur. Model checking parameterized systems. In *Handbook of Model Checking*, pages 685–725. Springer, 2018. doi:10.1007/978-3-319-10575-8_21.

[BF13] Nathalie Bertrand and Paulin Fournier. Parameterized verification of many identical probabilistic timed processes. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.501.

[BJK+15] Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. doi:10.1007/978-3-031-02011-7.

[BK00] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2), 2000. doi:10.1016/S0004-3702(99)00071-5.

[BM06] Jorge A. Baier and Sheila A. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. AAAI Press, 2006.

[Boj10] Mikolaj Bojanczyk. Beyond ω-regular languages. In *27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 11–16, 2010. doi:10.4230/LIPIcs.STACS.2010.2440.

[CETFX09] Remy Chevallier, Emmanuelle Encrenaz-Tiphene, Laurent Fribourg, and Weiwen Xu. Timed verification of the generic architecture of a memory circuit using parametric timed automata. *Formal Methods in System Design*, 34(1):59–81, 2009. doi:10.1007/s10703-008-0061-x.

[CTV08] Edmund Clarke, Murali Talupur, and Helmut Veith. Proving Ptolemy right: The environment abstraction framework for model checking concurrent systems. In *International Conference on*

Tools and Algorithms for the Construction and Analysis of Systems, pages 33–47. Springer, 2008. doi:10.1007/978-3-540-78800-3_4.

[DGV13] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[DSZ10] Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *International Conference on Concurrency Theory*, pages 313–327. Springer, 2010. doi:10.1007/978-3-642-15375-4_22.

[EFM99] Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Proceedings of 14th Symposium on Logic in Computer Science*, pages 352–359. IEEE, 1999. doi:10.1109/LICS.1999.782630.

[EL87] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987. doi:10.1016/0167-6423(87)90036-0.

[FH15] Estíbaliz Fraca and Serge Haddad. Complexity analysis of continuous Petri nets. *Fundamenta Informaticae*, 137(1):1–28, 2015. doi:10.3233/FI-2015-1168.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GS92] Steven M German and A Prasad Sistla. Reasoning about systems with many processes. *Journal of ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.

[HMP92] Thomas A Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 545–558. Springer, 1992. doi:10.1007/3-540-55719-9_103.

[HP79] John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

[Ise17] Tobias Isenberg. Incremental inductive verification of parameterized timed systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(2):1–24, 2017. doi:10.1145/2984640.

[Jon97] Neil D. Jones. *Computability and complexity - from a programming perspective*. Foundations of computing series. MIT Press, 1997. doi:10.7551/mitpress/2003.001.0001.

[JS18] Swen Jacobs and Mouhammad Sakr. Analyzing guarded protocols: Better cutoffs, more systems, more expressivity. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science, pages 247–268. Springer, 2018. doi:10.1007/978-3-319-73721-8_12.

[KKW14] Alexander Kaiser, Daniel Kroening, and Thomas Wahl. A widening approach to multithreaded program verification. *ACM Transactions on Programming Languages and Systems*, 36(4):14:1–14:29, 2014. doi:10.1145/2629608.

[KL16] Panagiotis Kouvaros and Alessio Lomuscio. Parameterised verification for multi-agent systems. *Artificial Intelligence*, 234:152–189, 2016. doi:10.1016/j.artint.2016.01.008.

[LP22] Alessio Lomuscio and Edoardo Pirovano. A counter abstraction technique for verifying properties of probabilistic swarm systems. *Artificial Intelligence*, 305:103666, 2022. doi:10.1016/j.artint.2022.103666.

[McM01] Kenneth L. McMillan. Parameterized verification of the FLASH cache coherence protocol by compositional model checking. In *Proceedings of Correct Hardware Design and Verification Methods, 11th IFIP WG 10.5 Advanced Research Working Conference, CHARME*, volume 2144 of *Lecture Notes in Computer Science*, pages 179–195. Springer, 2001. doi:10.1007/3-540-44798-9_17.

[OW03] Joël Ouaknine and James Worrell. Revisiting digitization, robustness, and decidability for timed automata. In *Proceedings of 18th Annual IEEE Symposium of Logic in Computer Science*, pages 198–207. IEEE, 2003. doi:10.1109/LICS.2003.1210059.

[Saf88] Shmuel Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science*, pages 319–327. IEEE, 1988. doi:10.1109/SFCS.1988.21948.

[SS14] Luca Spalazzi and Francesco Spegni. Parameterized model-checking of timed systems with conjunctive guards. In *Verified Software: Theories, Tools and Experiments*, Lecture Notes in Computer Science, pages 235–251. Springer, 2014. doi:10.1007/978-3-319-12154-3_15.

[SS20] Luca Spalazzi and Francesco Spegni. Parameterized model checking of networks of timed automata with boolean guards. *Theoretical Computer Science*, 813:248–269, 2020. doi:10.1016/j.tcs.2019.12.026.

[Suz88] Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Information Processing Letters*, 28(4):213–214, 1988. doi:10.1016/0020-0190(88)90211-6.

[Var95] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency - Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*. Springer, 1995. doi:10.1007/3-540-60915-6_6.

[ZP04] Lenore D. Zuck and Amir Pnueli. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems and Structures*, 30(3-4):139–169, 2004. doi:10.1016/j.cl.2004.02.006.