Model Checking Parameterized Asynchronous Shared-Memory Systems

Antoine Durand-Gasselin[1], Javier Esparza[1], Pierre Ganty[2(✉)], and Rupak Majumdar[3]

[1] TU Munich, Munich, Germany
[2] IMDEA Software Institute, Madrid, Spain
pierre.ganty@imdea.org
[3] MPI-SWS, Kaiserslautern, Germany

Abstract. We characterize the complexity of liveness verification for parameterized systems consisting of a leader process and arbitrarily many anonymous and identical contributor processes. Processes communicate through a shared, bounded-value register. While each operation on the register is atomic, there is no synchronization primitive to execute a sequence of operations atomically.

We analyze the case in which processes are modeled by finite-state machines or pushdown machines and the property is given by a Büchi automaton over the alphabet of read and write actions of the leader. We show that the problem is decidable, and has a surprisingly low complexity: it is NP-complete when all processes are finite-state machines, and is PSPACE-hard and in NEXPTIME when they are pushdown machines. This complexity is lower than for the non-parameterized case: liveness verification of finitely many finite-state machines is PSPACE-complete, and undecidable for two pushdown machines.

For finite-state machines, our proofs characterize infinite behaviors using existential abstraction and semilinear constraints. For pushdown machines, we show how contributor computations of high stack height can be simulated by computations of many contributors, each with low stack height. Together, our results characterize the complexity of verification for parameterized systems under the assumptions of anonymity and asynchrony.

1 Introduction

We study the verification problem for *parameterized asynchronous shared-memory systems* [9,12]. These systems consist of a *leader* process and arbitrarily many identical *contributors*, processes with no identity, running at arbitrary relative speeds. The shared-memory consists of a read/write register that all processes can access to perform either a read operation or a write operation. The register is bounded: the set of values that can be stored is finite. Read/write operations execute atomically but sequences of operations do not: no process can conduct an atomic sequence of reads and writes while excluding all other processes. In a previous paper [9], we have studied the complexity

D. Kroening and C.S. Păsăreanu (Eds.): CAV 2015, Part I, LNCS 9206, pp. 67–84, 2015.
DOI: 10.1007/978-3-319-21690-4_5

of safety verification, which asks to check if a safety property holds no matter how many contributors are present. In a nutshell, we showed that the problem is coNP-complete when both leader and contributors are finite-state automata and PSPACE-complete when they are pushdown automata.

In this paper we complete the study of this model by addressing the verification of liveness properties specified as ω-regular languages (which in particular encompasses LTL model-checking). Given a property like "every request is eventually granted" and a system with a fixed number of processes, one is often able to guess an upper bound on the maximal number of steps until the request is granted, and replace the property by the safety property "every request is granted after at most K steps." In parameterized systems this bound can depend on the (unbounded) number of processes, and so reducing liveness to safety, or to finitary reasoning, is not obvious. Indeed, for many parameterized models, liveness verification is undecidable even if safety is decidable [8,13].

Our results show that there is no large complexity gap between liveness and safety verification: liveness verification (existence of an infinite computation violating a property) is NP-complete in the finite-state case, and PSPACE-hard and in NEXPTIME in the pushdown case. In contrast, remember that liveness checking is already PSPACE-complete for a *finite* number of finite-state machines, and undecidable for a *fixed* number of pushdown systems. Thus, not only is liveness verification decidable in the parameterized setting but the complexity of the parameterized problem is *lower* than in the non-parameterized case, where all processes are part of the input. We interpret this as follows: in asynchronous shared-memory systems, the existence of arbitrarily many processes leads to a "noisy" environment, in which contributors may hinder progress by replying to past messages from the leader, long after the computation has moved forward to a new phase. It is known that imperfect communication can *reduce* the power of computation and the complexity of verification problems: the best known example are lossy channel systems, for which many verification problems are decidable, while they are undecidable for perfect channels (see e.g. [1,3]). Our results reveal another instance of the same phenomenon.

Technically, our proof methods are very different from those used for safety verification. Our previous results [9] relied on a fundamental Simulation Lemma, inspired by Hague's work [12], stating that the *finite* behaviors of an arbitrary number of contributors can be simulated by a finite number of *simulators*, one for each possible value of the register. Unfortunately, the Simulation Lemma does not extend to infinite behaviors, and so we have to develop new ideas. In the case in which both leader and contributors are finite-state machines, the NP-completeness result is obtained by means of a combination of an abstraction that overapproximates the set of possible infinite behaviors, and a semilinear constraint that allows us to regain precision. The case in which both leader and contributors are pushdown machines is very involved. In a nutshell, we show that pushdown runs in which a parameter called the *effective stack height* grows too much can be "distributed" into a number of runs with smaller effective stack height. We then prove that the behaviors of a pushdown machine with

a bounded effective stack height can be simulated by an exponentially larger finite-state machine.

Related Work. Parameterized verification has been studied extensively, both theoretically and practically. While very simple variants of the problem are already undecidable [6], many non-trivial parameterized models retain decidability. There is no clear "rule of thumb" that allows one to predict what model checking problems are decidable, nor their complexities, other than "liveness is generally harder than safety." For example, coverability for Petri nets—in which finite-state, identityless processes communicate via rendezvous or global shared state— is EXPSPACE-complete, higher than the PSPACE-completeness of the non-parameterized version, and verification of liveness properties can be equivalent to Petri net reachability, for which we only know non-primitive recursive upper bounds, or even undecidable. Safety verification for extensions to Petri nets with reset or transfer, or broadcast protocols, where arbitrarily many finite-state processes communicate through broadcast messages, are non-primitive recursive; liveness verification is undecidable in all cases [2,8,13]. Thus, our results, which show simultaneously lower complexity than non-parameterized problems, as well as similar complexity for liveness and safety, are quite unexpected.

German and Sistla [10] and Aminof *et al.* [4] have studied a parameterized model with rendezvous as communication primitive, where processes are finite-state machines. Model checking the fully symmetrical case—only contributors, no leaders—runs in polynomial time (other topologies have also been considered [4]), while the asymmetric case with a leader is EXPSPACE-complete. In this paper we study the same problems, but for a shared memory communication primitive.

Population protocols [5] are another well-studied model of identityless asynchronous finite-state systems communicating via rendezvous. The semantics of population protocols is given over fair runs, in which every potential interaction that is infinitely often enabled is infinitely often taken. With this semantics, population protocols compute exactly the semilinear predicates [5]. In this paper we do not study what our model can compute (in particular, we are agnostic with respect to which fairness assumptions are reasonable), but what we can compute or decide about the model.

2 Formal Model: Non-atomic Networks

In this paper, we identify systems with languages. System actions are modeled as symbols in an alphabet, executions are modeled as infinite words, and the system itself is modeled as the language of its executions. Composition operations that combine systems into larger ones are modeled as operations on languages.

2.1 Systems as Languages

An *alphabet* Σ is a finite, non-empty set of *symbols*. A *word* over Σ is a finite sequence over Σ including the empty sequence denoted ε, and a *language* is a

set of words. An *ω-word* over Σ is an infinite sequence of symbols of Σ, and an *ω-language* is a set of ω-words. We use Σ^* (resp. Σ^ω) to denote the language of all words (resp. ω-words) over Σ. When there is no ambiguity, we use "words" to refer to words or ω-words. We do similarly for languages. Let w be a sequence over some alphabet, define $\mathrm{dom}(w) = \{1, \ldots, n\}$ if $w = a_1 a_2 \ldots a_n$ is a word; else (w is an ω-word) $\mathrm{dom}(w)$ denote the set $\mathbb{N} \setminus \{0\}$. Elements of $\mathrm{dom}(w)$ are called *positions*. The *length* of a sequence w is defined to be $\sup \mathrm{dom}(w)$ and is denoted $|w|$. We denote by $(w)_i$ the symbol of w at position i if $i \in \mathrm{dom}(w)$, ε otherwise. Moreover, let $(w)_{i..j}$ with $i, j \in \mathbb{N}$ and $i < j$ denote $(w)_i(w)_{i+1} \ldots (w)_j$. Also $(w)_{i..\infty}$ denotes $(w)_i(w)_{i+1} \ldots$ For words $u, v \in (\Sigma^\omega \cup \Sigma^*)$, we say u is a *prefix* of v if either $u = v$ or $u \in \Sigma^*$ and there is a $w \in (\Sigma^\omega \cup \Sigma^*)$ such that $v = uw$.

Combining Systems: Shuffle. Intuitively, the shuffle of systems L_1 and L_2 is the system interleaving the executions of L_1 with those of L_2. Given two ω-languages $L_1 \subseteq \Sigma_1^\omega$ and $L_2 \subseteq \Sigma_2^\omega$, their *shuffle*, denoted by $L_1 \between L_2$, is the ω-language over $(\Sigma_1 \cup \Sigma_2)$ defined as follows. Given two ω-words $x \in \Sigma_1^\omega, y \in \Sigma_2^\omega$, we say that $z \in (\Sigma_1 \cup \Sigma_2)^\omega$ is an *interleaving* of x and y if there exist (possibly empty) words $x_1, x_2, \ldots, x_i, \ldots \in \Sigma_1^*$ and $y_1, y_2, \ldots, y_i, \ldots \in \Sigma_2^*$ such that each $x_1 x_2 \cdots x_i$ is a prefix of x, and each $y_1 y_2 \cdots y_i$ is a prefix of y, and $z = x_1 y_1 x_2 y_2 \cdots x_i y_i \cdots \in \Sigma^\omega$ is an ω-word. Then $L_1 \between L_2 = \bigcup_{x \in L_1, y \in L_2} x \between y$, where $x \between y$ denotes the set of all interleavings of x and y. For example, if $L_1 = ab^\omega$ and $L_2 = ab^\omega$, we get $L_1 \between L_2 = (a + ab^*a)b^\omega$. Shuffle is associative and commutative, and so we can write $L_1 \between \cdots \between L_n$ or $\between_{i=1}^n L_i$.

Combining Systems: Asynchronous product. The asynchronous product of $L_1 \subseteq \Sigma_1^\omega$ and $L_2 \subseteq \Sigma_2^\omega$ also interleaves the executions but, this time, the actions in the common alphabet must now be executed jointly. The ω-language of the resulting system, called the *asynchronous product* of L_1 and L_2, is denoted by $L_1 \parallel L_2$, and defined as follows. Let $Proj_\Sigma(w)$ be the word obtained by erasing from w all occurrences of symbols not in Σ. $L_1 \parallel L_2$ is the ω-language over the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ such that $w \in L_1 \parallel L_2$ iff $Proj_{\Sigma_1}(w)$ and $Proj_{\Sigma_2}(w)$ are prefixes of words in L_1 and L_2, respectively. We abuse notation and write $w_1 \parallel L_2$ instead of $\{w_1\} \parallel L_2$ when $L_1 = \{w_1\}$. For example, let $\Sigma_1 = \{a, c\}$ and $\Sigma_2 = \{b, c\}$. For $L_1 = (ac)^\omega$ and $L_2 = (bc)^\omega$ we get $L_1 \parallel L_2 = ((ab + ba)c)^\omega$. Observe that the language $L_1 \parallel L_2$ depends on L_1, L_2 and also on Σ_1 and Σ_2. For example, if $\Sigma_1 = \{a\}$ and $\Sigma_2 = \{b\}$, then $\{a^\omega\} \parallel \{b^\omega\} = (a + b)^\omega$, but if $\Sigma_1 = \{a, b\} = \Sigma_2$, then $\{a^\omega\} \parallel \{b^\omega\} = \emptyset$. So we should more properly write $L_1 \parallel_{\Sigma_1, \Sigma_2} L_2$. However, since the alphabets Σ_1 and Σ_2 will be clear from the context, we will omit them. Like shuffle, asynchronous product is also associative and commutative, and so we write $L_1 \parallel \cdots \parallel L_n$. Notice finally that shuffle and asynchronous product coincide if $\Sigma_1 \cap \Sigma_2 = \emptyset$, but usually differ otherwise. For instance, if $L_1 = ab^\omega$ and $L_2 = ab^\omega$, we get $L_1 \parallel L_2 = ab^\omega$.

We describe systems as combinations of shuffles and asynchronous products, for instance we write $L_1 \parallel (L_2 \between L_3)$. In these expressions we assume that $\between$ binds tighter than $\parallel$, and so $L_1 \between L_2 \parallel L_3$ is the language $(L_1 \between L_2) \parallel L_3$, and not $L_1 \between (L_2 \parallel L_3)$.

2.2 Non-atomic Networks

A non-atomic network is an infinite family of systems parameterized by a number k. The kth element of the family has $k+1$ components communicating through a global store by means of read and write actions. The store is modeled as an atomic register whose set of possible values is finite. One of the $k+1$ components is the leader, while the other k are the contributors. All contributors have exactly the same possible behaviors (they are copies of the same ω-language), while the leader may behave differently. The network is called non-atomic because components cannot atomically execute sequences of actions, only one single read or write.

Formally, we fix a finite set $\mathcal{G}$ of *global values*. A *read-write alphabet* is any set of the form $\mathcal{A} \times \mathcal{G}$, where $\mathcal{A}$ is a set of *read* and *write (actions)*. We denote a symbol $(a, g) \in \mathcal{A} \times \mathcal{G}$ by $a(g)$ and define $\mathcal{G}(a_1, \ldots, a_n) = \{a_i(g) \mid 1 \leq i \leq n,\ g \in \mathcal{G}\}$.

We fix two languages $\mathcal{D} \subseteq \Sigma_{\mathcal{D}}^{\omega}$ and $\mathcal{C} \subseteq \Sigma_{\mathcal{C}}^{\omega}$, called the *leader* and the *contributor*, with alphabets $\Sigma_{\mathcal{D}} = \mathcal{G}(r_d, w_d)$ and $\Sigma_{\mathcal{C}} = \mathcal{G}(r_c, w_c)$, respectively, where r_d, r_c are called *reads* and w_c, w_d are called *writes*. We write $w_\star$ (respectively, $r_\star$) to stand for either w_c or w_d (respectively, r_c or r_d). We further assume that $Proj_{\{r_\star(g), w_\star(g)\}}(\mathcal{D} \cup \mathcal{C}) \neq \emptyset$ holds for every $g \in \mathcal{G}$, else the value g is never used and can be removed from $\mathcal{G}$.

Additionally, we fix an ω-language $\mathcal{S}$, called the *store*, over $\Sigma_{\mathcal{D}} \cup \Sigma_{\mathcal{C}}$. It models the sequences of read and write operations supported by an atomic register: a write $w_\star(g)$ writes g to the register, while a read $r_\star(g)$ succeeds when the register's current value is g. Initially the store is only willing to execute a write. Formally $\mathcal{S}$ is defined as $\left(\sum_{g\in\mathcal{G}} \left(\ w_\star(g)\,(r_\star(g))^* \ \right)\right)^{\omega} + \left(\sum_{g\in\mathcal{G}} \left(\ w_\star(g)\,(r_\star(g))^* \ \right)^* \ \sum_{g\in\mathcal{G}} \left(\ w_\star(g)\,(r_\star(g))^{\omega} \ \right)\right)$ and any finite prefix thereof. Observe that $\mathcal{S}$ is completely determined by $\Sigma_{\mathcal{D}}$ and $\Sigma_{\mathcal{C}}$. Figure 1 depicts a store with $\{1, 2, 3\}$ as possible values as the language of a transition system.

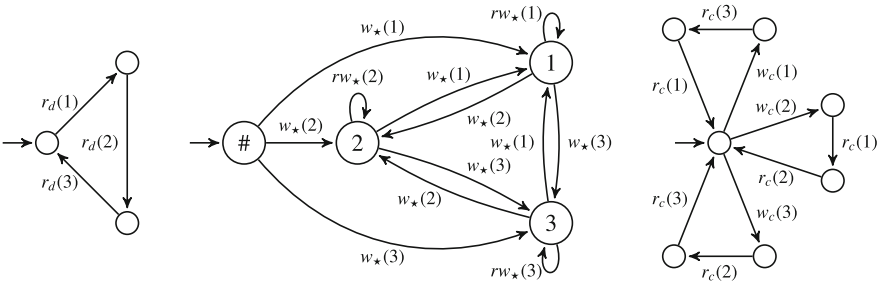

Fig. 1. Transition systems describing languages $\mathcal{D}$, $\mathcal{S}$, and $\mathcal{C}$. We write $rw_\star(g) = r_\star(g) \cup w_\star(g) = \{r_c(g), r_d(g)\} \cup \{w_c(g), w_d(g)\}$. The transition system for $\mathcal{S}$ is in state $i \in \{1, 2, 3\}$ when the current value of the store is i.

Definition 1. Let $\mathcal{D} \subseteq \Sigma_{\mathcal{D}}^{\omega}$ and $\mathcal{C} \subseteq \Sigma_{\mathcal{C}}^{\omega}$ be a leader and a contributor, and let $k \geq 1$. The *k-instance of the $(\mathcal{D},\mathcal{C})$-network* is the ω-language $\mathcal{N}^{(k)} = (\mathcal{D} \parallel \mathcal{S} \parallel \{\!|_k\, \mathcal{C})$ where $\{\!|_k \mathcal{C}$ stands for $\{\!|_{i=1}^{k}\, \mathcal{C}$. The $(\mathcal{D},\mathcal{C})$-network $\mathcal{N}$ is the ω-language $\mathcal{N} = \bigcup_{k=1}^{\infty} \mathcal{N}^{(k)}$. We omit the prefix $(\mathcal{D},\mathcal{C})$ when it is clear from the context. It follows easily from the properties of shuffle and asynchronous product that $\mathcal{N} = (\mathcal{D} \parallel \mathcal{S} \parallel \{\!|_\infty \mathcal{C})$, where $\{\!|_\infty \mathcal{C}$ is an abbreviation of $\bigcup_{k=1}^{\infty} \{\!|_k \mathcal{C}$.

Next we introduce a notion of *compatibility* between a word of the leader and a multiset of words of the contributor (a multiset because several contributors may execute the same sequence of actions). Intuitively, compatibility means that all the words can be interleaved into a legal infinite sequence of reads and writes supported by an atomic register—that is, an infinite sequence belonging to $\mathcal{S}$. Formally:

Definition 2. Let $u \in \Sigma_{\mathcal{D}}^{\omega}$, and let $M = \{v_1, \ldots, v_k\}$ be a multiset of words over $\Sigma_{\mathcal{C}}^{\omega}$ (possibly containing multiple copies of a word). We say that u is *compatible* with M iff the ω-language $(u \parallel \mathcal{S} \parallel \{\!|_{i=1}^{k}\, v_i)$ is non-empty. When u and M are compatible, there exists a word $s \in \mathcal{S}$ such that $(u \parallel s \parallel \{\!|_{i=1}^{k}\, v_i) \neq \emptyset$. We call s a *witness* of compatibility.

Example 1. Consider the network with $\mathcal{G} = \{1,2,3\}$ where the leader, store, and contributor languages are given by the infinite paths of the transition systems from Fig. 1. The only ω-word of $\mathcal{D}$ is $(r_d(1)r_d(2)r_d(3))^{\omega}$ and the ω-language of $\mathcal{C}$ is $(w_c(1)r_c(3)r_c(1) + w_c(2)r_c(1)r_c(2) + w_c(3)r_c(2)r_c(3))^{\omega}$. For instance, $\mathcal{D} = (r_d(1)r_d(2)r_d(3))^{\omega}$ is compatible with the multiset M of 6 ω-words obtained by taking two copies of $(w(1)r(3)r(1))^{\omega}$, $(w(2)r(1)r(2))^{\omega}$ and $(w(3)r(2)r(3))^{\omega}$. The reader may be interested in finding another multiset compatible with $\mathcal{D}$ and containing only 4 ω-words.

Stuttering Property. Intuitively, the stuttering property states that if we take an ω-word of a network $\mathcal{N}$ and "stutter" reads and writes of the contributors, going e.g. from $w_d(1)r_c(1)w_c(2)r_d(2)\ldots$ to $w_d(1)r_c(1)r_c(1)w_c(2)w_c(2)w_c(2)r_d(2)\ldots$, the result is again an ω-word of the network.

Let $s \in \mathcal{S}$ be a witness of compatibility of $u \in \Sigma_{\mathcal{D}}^{\omega}$ and $M = \{v_1, \ldots, v_k\}$. Pick a set I of positions (viz. $I \subseteq \mathrm{dom}(s)$) such that $(s)_i \in \Sigma_{\mathcal{C}}$ for each $i \in I$, and pick a number $\ell_i \geq 0$ for every $i \in I$. Let s' be the result of simultaneously replacing each $(s)_i$ by $(s)_i^{\ell_i+1}$ in s. We have that $s' \in \mathcal{S}$. Now let $v_s = (s)_{i_1}^{\ell_{i_1}} \cdot (s)_{i_2}^{\ell_{i_2}} \cdots$, where $i_1 = \min(I)$, $i_2 = \min(I \setminus \{i_1\})$, ... It is easy to see that $(u \parallel s' \parallel v_s \{\!|\, \{\!|_{i=1}^{k}\, v_i) \neq \emptyset$, and so u is compatible with $M \oplus \{v_s\}$, the multiset consisting of M and v_s, and s' is a witness of compatibility.

An easy consequence of the stuttering property is the *copycat lemma* [9].

Lemma 1. (Copycat Lemma). *Let $u \in \Sigma_{\mathcal{D}}^{\omega}$ and let M be a multiset of words of $\Sigma_{\mathcal{C}}^{\omega}$. If u is compatible with M, then u is also compatible with $M \oplus \{v\}$ for every $v \in M$.*

2.3 The Model-Checking Problem for Linear-Time Properties

We consider the model checking problem for linear-time properties, that asks, given a network $\mathcal{N}$ and an ω-regular language L, decide whether $\mathcal{N} \parallel L$ is non-empty. We assume L is given as a Büchi automaton A over $\Sigma_{\mathcal{D}}$. Intuitively, A is a tester that observes the actions of the leader; we call this the *leader model checking problem*.

We study the complexity of leader model checking for networks in which the read-write ω-languages $\mathcal{D}$ and $\mathcal{C}$ of leader and contributor are generated by an abstract machine, like a finite-state machine (FSM) or a pushdown machine (PDM). (We give formal definitions later.) More precisely, given two classes of machines $\mathtt{D}$, $\mathtt{C}$, we study the model checking problem $\mathtt{MC}(\mathtt{D}, \mathtt{C})$ defined as follows:

Given: machines $D \in \mathtt{D}$ and $C \in \mathtt{C}$, and a Büchi automaton A
Decide: Is $\mathcal{N}_A = (L(A) \parallel L(D) \parallel \mathcal{S} \parallel \mathbb{0}_\infty L(C))$ non-empty?

In the next sections we prove that $\mathtt{MC}(\mathtt{FSM},\mathtt{FSM})$ and $\mathtt{MC}(\mathtt{PDM},\mathtt{FSM})$ are NP-complete, while $\mathtt{MC}(\mathtt{PDM},\mathtt{PDM})$ is in NEXPTIME and PSPACE-hard.

Example 2. Consider the instance of the model checking problem where $\mathcal{D}$ and $\mathcal{C}$ are as in Fig. 1, and A is a Büchi automaton recognizing all words over $\Sigma_{\mathcal{D}}$ containing infinitely many occurrences of $r_d(1)$. Since $\mathcal{D}$ is compatible with a multiset of words of the contributors, $\mathcal{N}_A$ is non-empty. In particular, $\mathcal{N}_A^{(4)} \neq \emptyset$.

Since $\Sigma_A = \Sigma_{\mathcal{D}}$, we can replace A and D by a machine $A \times D$ with a Büchi acceptance condition. The construction of $A \times D$ given A and D is standard. In what follows, we assume that D comes with a Büchi acceptance condition and forget about A.

There are two natural variants of the model checking problem, where $\Sigma_A = \Sigma_{\mathcal{C}}$, i.e., the alphabet of A contains the actions of all contributors, or $\Sigma_A = \Sigma_{\mathcal{D}} \cup \Sigma_{\mathcal{C}}$. In both these variants, the automaton A can be used to simulate atomic networks. Indeed, if the language of A consists of all sequences of the form $(w_d()r_c()w_c()r_d())^\omega$, and we design the contributors so that they alternate reads and writes, then the accepting executions are those in which the contributors read a value from the store and write a new value in an atomic step. So the complexity of the model-checking problem coincides with the complexity for atomic networks (undecidable for PDMs and EXPSPACE-complete for FSMs), and we do not study it further.

3 $\mathtt{MC}(\mathtt{FSM},\mathtt{FSM})$ is NP-Complete

We fix some notations. A finite-state machine (FSM) (Q, δ, q_0) over Σ consists of a finite set of states Q containing an initial state q_0 and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. A word $v \in \Sigma^\omega$ is *accepted* by an FSM if there exists a sequence $q_1 q_2 \cdots$ of states such that $(q_i, (v)_{i+1}, q_{i+1}) \in \delta$ for all $i \geq 0$. We denote by $q_0 \xrightarrow{(v)_1} q_1 \xrightarrow{(v)_2} \cdots$ the *run* accepting v. A Büchi automaton (Q, δ, q_0, F) is

an FSM (Q, δ, q_0) together with a set $F \subseteq Q$ of accepting states. An ω-word $v \in \Sigma^\omega$ is accepted by a Büchi automaton if there is a run $q_0 \xrightarrow{(v)_1} q_1 \xrightarrow{(v)_2} \cdots$ such that $q_j \in F$ for infinitely many positions j. The ω-language of a FSM or Büchi automaton A, denoted by $L(A)$, is the set of ω-words accepted by A.

In the rest of the section we show that MC(FSM,FSM) is NP-complete. Section 3.1 defines the infinite transition system associated to a (FSM,FSM)-network. Section 3.2 introduces an associated finite abstract transition system. Section 3.3 states and proves a lemma (Lemma 3) characterizing the cycles of the abstract transition system that, loosely speaking, can be concretized into infinite executions of the concrete transition system. Membership in NP is then proved using the lemma. NP-hardness follows from NP-hardness of reachability [9].

3.1 (FSM,FSM)-Networks: Populations and Transition System

We fix a Büchi automaton $D = (Q_D, \delta_D, q_{0D}, F)$ over $\Sigma_\mathcal{D}$ and an FSM $C = (Q_C, \delta_C, q_{0C})$ over $\Sigma_\mathcal{C}$. A *configuration* is a tuple $(q_D, g, \boldsymbol{p})$, where $q_D \in Q_D$, $g \in \mathcal{G} \cup \{\#\}$, and $\boldsymbol{p}\colon Q_C \to \mathbb{N}$ assigns to each state of C a natural number. Intuitively, q_D is the current state of D; g is a value or the special value $\#$, modelling that the store has not been initialized yet, and no process read before some process writes; finally, $\boldsymbol{p}(q)$ is the number of contributors currently at state $q \in Q_C$. We call $\boldsymbol{p}$ a *population* of Q_C, and write $|\boldsymbol{p}| = \sum_{q \in Q_C} \boldsymbol{p}(q)$ for the *size* of $\boldsymbol{p}$. Linear combinations of populations are defined componentwise: for every state $q \in Q_C$, we have $(k_1\boldsymbol{p_1} + k_2\boldsymbol{p_2})(q) := k_1\boldsymbol{p_1}(q) + k_2\boldsymbol{p_2}(q)$. Further, given $q \in Q_C$, we denote by $\boldsymbol{q}$ the population $\boldsymbol{q}(q') = 1$ if $q = q'$ and $\boldsymbol{q}(q') = 0$ otherwise, i.e., the population with one contributor in state q and no contributors elsewhere. A configuration is *accepting* if the state of D is accepting, that is whenever $q_D \in F$. Given a set of populations $\boldsymbol{P}$, we define $(q_D, g, \boldsymbol{P}) := \{(q_D, g, \boldsymbol{p}) \mid \boldsymbol{p} \in \boldsymbol{P}\}$.

The labelled transition system $TS = (X, T, X_0)$ associated to $\mathcal{N}_A$ is defined as follows:

- X is the set of all configurations, and $X_0 \subseteq X$ is the set of initial configurations, given by $(q_{0D}, \#, \boldsymbol{P_0})$, where $\boldsymbol{P_0} = \{k\boldsymbol{q_{0C}} \mid k \geq 1\}$;
- $T = T_D \cup T_C$, where
  - T_D is the set of triples $\big((q_D, g, \boldsymbol{p})\,,\, t\,,\, (q'_D, g', \boldsymbol{p})\big)$ such that t is a transition of D, viz. $t \in \delta_D$, and one of the following conditions holds: (i) $t = (q_D, w_d(g'), q'_D)$; or (ii)$t = (q_D, r_d(g), q'_D)$, $g = g'$.
  - T_C is the set of triples $\big((q_D, g, \boldsymbol{p})\,,\, t\,,\, (q_D, g', \boldsymbol{p'})\big)$ such that $t \in \delta_C$, and one of the following conditions holds: (iii) $t = (q_C, w_c(g'), q'_C)$, $\boldsymbol{p} \geq \boldsymbol{q_C}$, and $\boldsymbol{p'} = \boldsymbol{p} - \boldsymbol{q_C} + \boldsymbol{q'_C}$; or (iv) $t = (q_C, r_c(g), q'_C)$, $\boldsymbol{p} \geq \boldsymbol{q_C}$, $g = g'$, and $\boldsymbol{p'} = \boldsymbol{p} - \boldsymbol{q_C} + \boldsymbol{q'_C}$.

  Observe that $|\boldsymbol{p}| = |\boldsymbol{p'}|$, because the total number of contributors of a population remains constant. Given configurations c and c', we write $c \xrightarrow{t} c'$ if $(c, t, c') \in T$.

We introduce a notation important for Lemma 3 below. We define $\Delta(t) := \boldsymbol{p'} - \boldsymbol{p}$. Observe that $\Delta(t) = \boldsymbol{0}$ in cases (i) and (ii) above, and $\Delta(t) = -\boldsymbol{q_C} + \boldsymbol{q_C}'$ in cases (iii) and (iv). So $\Delta(t)$ depends only on the transition t, but not on $\boldsymbol{p}$.

3.2 The Abstract Transition System

We introduce an *abstraction function* α that assigns to a set $\boldsymbol{P}$ of populations the set of states of Q_C populated by $\boldsymbol{P}$. We also introduce a *concretization function* γ that assigns to a set $Q \subseteq Q_C$ the set of all populations $\boldsymbol{p}$ that only populate states of Q. Formally:

$$\alpha(\boldsymbol{P}) = \{q \in Q_C \mid \boldsymbol{p}(q) \geq 1 \text{ for some } \boldsymbol{p} \in \boldsymbol{P}\}$$
$$\gamma(Q) = \{\boldsymbol{p} \mid \boldsymbol{p}(q) = 0 \text{ for every } q \in Q_C \setminus Q\} \ .$$

It is easy to see that α and γ satisfy $\gamma(\alpha(\boldsymbol{P})) \supseteq \boldsymbol{P}$ and $\alpha(\gamma(Q)) = Q$, and so α and γ form a Galois connection (actually, a Galois insertion). An *abstract configuration* is a tuple (q_D, g, Q), where $q_D \in Q_D$, $g \in \mathcal{G} \cup \{\#\}$, and $Q \subseteq Q_C$. We extend α and γ to (abstract) configurations in the obvious way. An abstract configuration is *accepting* when the state of D is accepting, that is whenever $q_D \in F$.

Given $TS = (X, T, X_0)$, we define its *abstraction* $\alpha TS = (\alpha X, \alpha T, \alpha X_0)$ as follows:

- $\alpha X = Q_D \times (\mathcal{G} \cup \{\#\}) \times 2^{Q_C}$ is the set of all abstract configurations.
- $\alpha X_0 = (q_{0D}, \#, \alpha(\boldsymbol{P_0})) = (q_{0D}, \#, \{q_{0C}\})$ is the initial configuration.
- $(\ (q_D, g, Q),\ t,\ (q'_D, g', Q')\ ) \in \alpha T$ iff there is $\boldsymbol{p} \in \gamma(Q)$ and $\boldsymbol{p}'$ such that $(q_D, g, \boldsymbol{p}) \xrightarrow{t} (q'_D, g', \boldsymbol{p}')$ and $Q' = \alpha(\{\boldsymbol{p}' \mid \exists \boldsymbol{p} \in \gamma(Q) \colon (q_D, g, \boldsymbol{p}) \xrightarrow{t} (q'_D, g', \boldsymbol{p}')\})$.

Observe that the number of abstract configurations is bounded by $K = |Q_D| \cdot |\mathcal{G}| + 1 \cdot 2^{|Q_C|}$. Let us point out that our abstract transition system resembles but is different from that of Pnueli et al. [14]. We write $a \xrightarrow{t}_\alpha a'$ if $(a, t, a') \in \alpha T$. The abstraction satisfies the following properties:

(A) For each ω-path $c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} c_2 \cdots$ of TS, there exists an ω-path $a_0 \xrightarrow{t_1}_\alpha a_1 \xrightarrow{t_2}_\alpha a_2 \cdots$ in αTS such that $c_i \in \gamma(a_i)$ for all $i \geq 0$.

(B) If $(q_D, g, Q) \xrightarrow{t}_\alpha (q'_D, g', Q')$, then $Q \subseteq Q'$.
To prove this claim, consider two cases:
- $t \in \delta_D$. Then $(q_D, g, \boldsymbol{p}) \xrightarrow{t} (q'_D, g', \boldsymbol{p})$ for every population $\boldsymbol{p}$ (because only the leader moves). So $(q_D, g, Q) \xrightarrow{t}_\alpha (q'_D, g', Q)$.
- $t \in \delta_C$. Consider the population $\boldsymbol{p} = 2\sum_{q \in Q} \boldsymbol{q} \in \gamma(Q)$. Then $(q_D, g, \boldsymbol{p}) \xrightarrow{t} (q_D, g', \boldsymbol{p}')$, where $\boldsymbol{p}' = \boldsymbol{p} - \boldsymbol{q_C} + \boldsymbol{q_C}'$. But then $\boldsymbol{p}' \geq \sum_{q \in Q} \boldsymbol{q}$, and so $\alpha(\{\boldsymbol{p}'\}) \supseteq Q$, which implies $(q_D, g, Q) \xrightarrow{t}_\alpha (q_D, g', Q')$ for some $Q' \supseteq Q$.

So in every ω-path $a_0 \xrightarrow{t_1}_\alpha a_1 \xrightarrow{t_2}_\alpha a_2 \cdots$ of αTS, where $a_i = (q_{Di}, g_i, Q_i)$, there is an index i at which the Q_i stabilize, that is, $Q_i = Q_{i+k}$ holds for every $k \geq 0$. However, the converse of (A) does not hold: given a path $a_0 \xrightarrow{t_1}_\alpha a_1 \xrightarrow{t_2}_\alpha a_2 \cdots$ of αTS, there may be no path $c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} c_2 \cdots$ in TS such that $c_i \in \gamma(a_i)$ for every $i \geq 0$. Consider a contributor machine C with two states q_0, q_1 and

one single transition $t = (q_0, w_c(1), q_1)$. Then αTS contains the infinite path (omitting the state of the leader, which plays no role):

$$(\#, \{q_0\}) \xrightarrow{t}_\alpha (1, \{q_0, q_1\}) \xrightarrow{t}_\alpha (1, \{q_0, q_1\}) \xrightarrow{t}_\alpha (1, \{q_0, q_1\}) \cdots$$

However, the transitions of TS are of the form $(1, k_0\boldsymbol{q_0} + k_1\boldsymbol{q_1}) \xrightarrow{t} (1, (k_0{-}1)\boldsymbol{q_0} + (k_1{+}1)\boldsymbol{q_1})$, and so TS has no infinite paths.

3.3 Realizable Cycles of the Abstract Transition System

We show that the existence of an infinite accepting path in TS reduces to the existence of a certain lasso path in αTS. A lasso path consists of a stem and a cycle. Lemma 2 shows how every abstract finite path (like the stem) has a counterpart in TS. Lemma 3 characterizes precisely those cycles in αTS which have an infinite path counterpart in TS.

Lemma 2. *Let (q_D, g, Q) be an abstract configuration of αTS reachable from $(q_{0D}, \#, \alpha(\boldsymbol{P_0}))$ $(= \alpha X_0)$. For every $\boldsymbol{p} \in \gamma(Q)$, there exists $\hat{\boldsymbol{p}}$ such that $(q_D, g, \hat{\boldsymbol{p}})$ is reachable from $(q_{0D}, \#, \boldsymbol{P_0})$ and $\hat{\boldsymbol{p}} \geq \boldsymbol{p}$.*

Lemma 2 does not hold for atomic networks. Indeed, consider a contributor with transitions $q_0 \xrightarrow{w_c(1)} q_1 \xrightarrow{r_c(1):w_c(2)} q_2 \xrightarrow{r_c(2):w_c(3)} q_3$, where $r_c(i) : w_c(j)$ denotes that the read and the write happen in one single atomic step. Then we have (omitting the state of the leader, which does not play any rôle here):

$$(\#, \{q_0\}) \xrightarrow{w_c(1)}_\alpha (1, \{q_0, q_1\}) \xrightarrow{r_c(1):w_c(2)}_\alpha (2, \{q_0, q_1, q_2\}) \xrightarrow{r_c(2):w_c(3)}_\alpha (3, \{q_0, \ldots, q_3\}) \ .$$

Let $\boldsymbol{p}$ be the population putting one contributor in each of $q_0, \ldots, q_3$. This population belongs to $\gamma(\{q_0, \ldots, q_3\})$ but no configuration $(3, \hat{\boldsymbol{p}})$ with $\hat{\boldsymbol{p}} > \boldsymbol{p}$ is reachable from any population that only puts contributors in q_0, no matter how many. Indeed, after the first contributor moves to q_2, no further contributor can follow, and so we cannot have contributors simultaneously in both q_2 and q_3. On the contrary, in non-atomic networks the Copycat Lemma states that what the move by one contributor can always be replicated by arbitrarily many.

We proceed to characterized the cycles of the abstract transition system that can be "concretized". A *cycle* of αTS is a path $a_0 \xrightarrow{t_1}_\alpha a_1 \xrightarrow{t_2}_\alpha a_2 \cdots \xrightarrow{t_{n-1}}_\alpha a_n$ such that $a_n = a_0$. A cycle is *realizable* if there is an infinite path $c_0 \xrightarrow{t'_1} c_1 \xrightarrow{t'_2} c_2 \cdots$ of TS such that $c_k \in \gamma(a_{(k \mod n)})$ and $t'_{k+1} = t_{(k+1 \mod n)}$ for every $k \geq 0$.

Lemma 3. *A cycle $a_0 \xrightarrow{t_1}_\alpha a_1 \xrightarrow{t_2}_\alpha a_2 \cdots \xrightarrow{t_n}_\alpha a_n$ of αTS is realizable iff $\sum_{i=1}^{n} \Delta(t_i) = \mathbf{0}$.*

Theorem 1. *MC(FSM,FSM) is NP-complete.*

Proof. NP-hardness follows from the NP-hardness of reachability [9]. We show membership in NP with the following high-level nondeterministic algorithm whose correctness relies on Lemmas 2 and 3:

1. Guess a sequence $Q_1, \ldots, Q_\ell$ of subsets of Q_C such that $Q_i \subsetneq Q_{i+1}$ for all i, $0 < i < \ell$. Note that $\ell \leq |Q_C|$.
2. Compute the set $\mathcal{Q} = Q_D \times (\mathcal{G} \cup \{\#\}) \times \{\{q_{0C}\}, Q_1, \ldots, Q_\ell\}$ of abstract configurations and the set $\mathcal{T}$ of abstract transitions between configurations of $\mathcal{Q}$.
3. Guess an accepting abstract configuration $a \in \mathcal{Q}$, that is, an $a = (q_D, g, Q)$ such that q_D is accepting in D.
4. Check that a is reachable from the initial abstract configuration $(q_{0D}, \#, \{q_{0C}\})$ by means of abstract transitions of $\mathcal{T}$.
5. Check that the transition system with $\mathcal{Q}$ and $\mathcal{T}$ as states and transitions contains a cycle $a_0 \xrightarrow{t_1}_\alpha a_1 \cdots a_{n-1} \xrightarrow{t_n}_\alpha a_n$ such that $n \geq 1$, $a_0 = a_n = a$ and $\sum_{i=1}^{n} \Delta(t_i) = \mathbf{0}$.

We show that the algorithm runs in polynomial time. First, because the sequence guessed is no longer than $|Q_C|$, the guess can be done in polynomial time. Next, we give a polynomial algorithm for step (5):

- Compute an FSA[1] $A_a^{\circlearrowleft}$ over the alphabet $\delta_D \cup \delta_C$ with $\mathcal{Q}$ as set of states, $\mathcal{T}$ as set of transitions, a as initial state, and $\{a\}$ as set of final states.
- Use the polynomial construction of Seidl *et al.* [15] to compute an (existential) Presburger formula Ω for the Parikh image of $L(A_a^{\circlearrowleft})$. The free variables of Ω are in one-to-one correspondence with the transitions of $\delta_D \cup \delta_C$. Denote by x_t the variable corresponding to transition $t \in \delta_D \cup \delta_C$.
- Compute the formula

$$\Omega' = \Omega \wedge \bigwedge_{q_c \in Q_c} \left(\sum_{tgt(t)=q_c} x_t = \sum_{src(t)=q_c} x_t \right) \wedge \sum_{t \in \delta_D \cup \delta_C} x_t > 0$$

  where *tgt* and *src* returns the target and source states of the transition passed in argument. Ω' adds to Ω the realizability condition of Lemma 3.
- Check satisfiability of Ω'. This step requires nondeterministic polynomial time because satisfiability of an existential Presburger formula is in NP [11]. □

4 MC(PDM,FSM) is NP-Complete

A pushdown system (PDM) $P = (Q, \Gamma, \delta, q_0)$ over Σ consists of a finite set Q of states including the initial state q_0, a *stack alphabet* Γ including the bottom stack symbol $\bot$, and a set of *rules* $\delta \subseteq Q \times \Sigma \times \Gamma \times Q \times (\Gamma \backslash \{\bot\} \cup \{\mathsf{pop}\})$ which either push or pop as explained below. A *PDM-configuration* qw consists of a state $q \in Q$ and a word $w \in \Gamma^*$ (denoting the stack content). For $q, q' \in Q$, $a \in \Sigma$, $\gamma, \gamma' \in \Gamma$, $w, w' \in \Gamma^*$, we say a PDM-configuration $q'w$ (resp. $q'\gamma'\gamma w$) a-follows $q\gamma w$ if $(q, a, \gamma, q', \mathsf{pop}) \in \delta$, (resp. $(q, a, \gamma, q', \gamma') \in \delta$); we write $qw \xrightarrow{a} q'w'$ if $q'w'$ a-follows qw, and call it a *transition*. A *run* $c_0 \xrightarrow{(v)_1} c_1 \xrightarrow{(v)_2} \ldots$ on a word $v \in \Sigma^\omega$ is a sequence of PDM-configurations such that $c_0 = q_0\bot$ and

[1] A finite-state automaton (FSA) is an FSM which decides languages of finite words. Therefore an FSA is an FSM with a set F of accepting states.

$c_i \xrightarrow{(v)_{i+1}} c_{i+1}$ for all $i \geq 0$. We write $c \xrightarrow{*} c'$ if there is a run from c to c'. The language $L(P)$ of P is the set of all words $v \in \Sigma^\omega$ such that P has a run on v.

A Büchi PDM is a PDM with a set $F \subseteq Q$ of accepting states. A word is accepted by a Büchi PDM if there is a run on the word for which some state in F occurs infinitely often along the PDM-configurations. The following lemma characterizes accepting runs.

Lemma 4. *[7] Let c be a configuration. There is an accepting run starting from c if there are states $q \in Q$, $q_f \in F$, a stack symbol $\gamma \in \Gamma$ such that $c \xrightarrow{*} q\gamma w$ for some $w \in \Gamma^*$ and $q\gamma \xrightarrow{*} q_f u \xrightarrow{*} q\gamma w'$ for some $u, w' \in \Gamma^*$.*

We now show MC(PDM, FSM) is decidable, generalizing the proof from Sect. 3. Fix a Büchi PDM $P = (Q_D, \Gamma_D, \delta_D, q_{0D}, F)$, and a FSM $C = (Q_C, \delta_C, q_{0C})$. A *configuration* is a tuple $(q_D, w, g, \boldsymbol{p})$, where $q_D \in Q_D$, $w \in \Gamma_D^*$ is the stack content, $g \in \mathcal{G} \cup \{\#\}$, and $\boldsymbol{p}$ is a population. Intuitively, $q_D w$ is the PDM-configuration of the leader. We extend the definitions from Sect. 3 like accepting configuration in the obvious way.

We define a labeled transition system $TS = (X, T, X_0)$, where X is the set of configurations including the set $X_0 = (q_{0D}, \bot, \#, \boldsymbol{P_0})$ of initial configurations, and the transition relation $T = T_D \cup T_C$, where T_C is as before and T_D is the set of triples $\big((q_D, w, g, \boldsymbol{p}), t, (q'_D, w', g', \boldsymbol{p})\big)$ such that t is a transition (not a rule) of D, and one of the following conditions holds: (i) $t = (q_D w \xrightarrow{w_d(g')} q'_D w')$; or (ii) $t = (q_D w \xrightarrow{r_d(g)} q'_D w')$ and $g = g'$. We define the abstraction αTS of TS as the obvious generalization of the abstraction in Sect. 3. An accepting path of the (abstract) transition system is an infinite path with infinitely many accepting (abstract) configurations. As for MC(FSM, FSM), not every accepting path of the abstract admits a concretization, but we find a realizability condition in terms of linear constraints. Here we use again the polynomial construction of Seidl *et al.* [15] mentioned in the proof of Theorem 1, this time to compute an (existential) Presburger formula for the Parikh image of a pushdown automaton.

Theorem 2. *MC(PDM,FSM) is NP-complete.*

5 MC(PDM,PDM) is in NEXPTIME

We show how to reduce MC(PDM,PDM) to MC(PDM,FSM). We first introduce the notion of *effective stack height* of a PDM-configuration in a run of a PDM, and define, given a PDM C, an FSM C_k that simulates all the runs of C of effective stack height k. Then we show that, for $k \in O(n^3)$, where n is the size of C, the language $(L(D) \parallel \mathcal{S} \parallel \langle\!\rangle_\infty L(C))$ is empty iff $(L(D) \parallel \mathcal{S} \parallel \langle\!\rangle_\infty L(C_k))$ is empty.

5.1 A FSM for Runs of Bounded Effective Stack Height

Consider a run of a PDM that repeatedly pushes symbol on the stack. The stack height of the configurations[2] is unbounded, but, intuitively, the PDM only uses

[2] For readability, we write "configuration" for "PDM-configuration.".

the topmost stack symbol during the run. To account for this we define the notion of effective stack height.

Definition 3. Let $\rho = c_0 \xrightarrow{(v)_1} c_1 \xrightarrow{(v)_2} \cdots$ be an infinite run of a PDM on ω-word v, where $c_i = q_i w_i$. The *dark suffix* of c_i in ρ, denoted by $ds(w_i)$, is the longest suffix of w_i that is also a proper suffix of w_{i+k} for every $k \geq 0$. The *active prefix* $ap(w_i)$ of w_i is the prefix satisfying $w_i = ap(w_i) \cdot ds(w_i)$. The *effective stack height* of c_i in ρ is $|ap(w_i)|$. We say that ρ is *effectively k-bounded* (or simply k-bounded for the sake of readability) if every configuration of ρ has an effective stack height of at most k. Further, we say that ρ is *bounded* if it is k-bounded for some $k \in \mathbb{N}$. Finally, an ω-word of the PDM is *k-bounded*, respectively bounded, if it is the word generated by some k-bounded, respectively bounded, run (other runs for the same word may not be bounded).

Intuitively, the effective stack height measures the actual memory required by the PDM to perform its run. For example, repeatedly pushing symbols on the stack produces a run with effective stack height 1. Given a position in the run, the elements of the stack that are never popped are those in the longest common suffix of all subsequent stacks. The first element of that suffix may be read, therefore only the longest *proper* suffix is effectively useless, so no configuration along an infinite run has effective stack height 0.

Proposition 1. *Every infinite run of a PDM contains infinitely many positions at which the effective stack height is 1.*

Proof. Let $p_0 w_0 \rightarrow p_1 w_1 \rightarrow p_2 w_2 \rightarrow \cdots$ be any infinite run. Notice that $|w_i| \geq 1$ for every $i \geq 0$, because otherwise the run would not be infinite. Let X be the set of positions of the run defined as: $i \in X$ iff $|w_i| \leq |w_j|$ for every $j > i$. Observe that X is infinite, because the first configuration of minimal stack height, say $p_k w_k$ belongs to it, and so does the first configuration of minimal stack height of the suffix $p_{k+1} w_{k+1} \rightarrow \cdots$, etc. By construction, the configuration at every position in X has effective stack height 1. □

In a k-bounded run, whenever the stack height exceeds k, the $k+1$-th stack symbol will never become the top symbol again, and so it becomes useless. So, we can construct a finite-state machine P_k recognizing the words of $L(P)$ accepted by k-bounded runs.

Definition 4. Given a PDM $P = (Q, \Gamma, \delta, q_0)$, the FSM $P_k = (Q_k, \delta_k, q_{0k})$, called the k-restriction of Ps, is defined as follows: (a) $Q_k = Q \times \bigcup_{i=1}^{k} \Gamma^i$ (a state of P_k consists of a state of P and a stack content no longer than k); (b) $q_{0k} = (q_0, \bot)$; (c) δ_k contains a transition $(q, (w)_{1..k}) \xrightarrow{a} (q', (w')_{1..k})$ iff $qw \xrightarrow{a} q'w'$ is a transition (not a rule) of P.

Theorem 3. *Given a PDM P, w admits a k-bounded run in P iff $w \in L(P_k)$.*

5.2 The Reduction Theorem

We fix a Büchi PDM D and a PDM C. By Theorem 3, in order to reduce MC(PDM,PDM) to MC(PDM,FSM) it suffices to prove the following Reduction Theorem:

Theorem 4. (Reduction Theorem). *Let $N = 2|Q_C|^2|\Gamma_C| + 1$, where Q_C and Γ_C are the states and stack alphabet of C, respectively. Let C_N be the N-restriction of C. We have:*

$$\big(L(D) \parallel \mathcal{S} \parallel \langle\!\rangle_\infty L(C)\big) \neq \emptyset \quad \textit{iff} \quad \big(L(D) \parallel \mathcal{S} \parallel \langle\!\rangle_\infty L(C_N)\big) \neq \emptyset \ . \qquad (\dagger)$$

There are PDMs D, C for which (†) holds only for $N \in \Omega(|Q_C|^2|\Gamma_C|)$.

Theorems 4 and 2 provide an upper bound for MC(PDM,PDM). PSPACE-hardness of the reachability problem [9] gives a lower bound.

Theorem 5. *MC(PDM,PDM) is in NEXPTIME and PSPACE-hard. If the contributor is a one counter machine (with zero-test), it is NP-complete.*

The proof of Theorem 4 is very involved. Given a run of D compatible with a finite multiset of runs of C, we construct another run of D compatible with a finite multiset of N-bounded runs of C_N. (Here we extend compatibility to runs: runs are compatible if the words they accept are compatible.)

The proof starts with the Distributing lemma, which, loosely speaking, shows how to replace a run of C by a multiset of "smaller" runs of C without the leader "noticing". After this preliminary result, the first key proof element is the Boundedness Lemma. Let σ be an infinite run of D compatible with a finite multiset R of runs of C. The Boundedness Lemma states that, for any number Z, the first Z steps of σ are compatible with a (possibly larger) multiset R_Z of runs of C_N. Since the size of R_Z may grow with Z, this lemma does not yet prove Theorem 4: it only shows that σ is compatible with an *infinite* multiset of runs of C_N. This obstacle is overcome in the final step of the proof. We show that, for a sufficiently large Z, there are indices $i < j$ such that, not σ itself, but the run $(\sigma)_{1..i}\big((\sigma)_{i+1..j}\big)^\omega$ for adequate i and j is compatible with a *finite* multiset of runs of C_N. Loosely speaking, this requires to prove not only that the leader can repeat $(\sigma)_{i+1..j}$ infinitely often, but also that the runs executed by the instances of C_N while the leader executes $(\sigma)_{i+1..j}$ can be repeated infinitely often.

The Distributing Lemma. Let $\rho = c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} c_2 \xrightarrow{a_3} \cdots$ be a (finite or infinite) run of C. Let r_i be the PDM-rule of C generating the transition $c_{i-1} \xrightarrow{a_i} c_i$. Then ρ is completely determined by c_0 and the sequence $r_1 r_2 r_3 \ldots$ Since c_0 is also fixed (for fixed C), in the rest of the paper we also sometimes write $\rho = r_1 r_2 r_3 \ldots$ This notation allows us to speak of $\mathrm{dom}(\rho)$, $(\rho)_k$, $(\rho)_{i..j}$ and $(\rho)_{i..\infty}$.

We say that ρ *distributes to a multiset R of runs of C* if there exists an *embedding function* ψ that assigns to each run $\rho' \in R$ and to each position $i \in \mathrm{dom}(\rho')$ a position $\psi(\rho', i) \in \mathrm{dom}(\rho)$, and satisfies the following properties:

- $(\rho')_i = (\rho)_{\psi(\rho',i)}$. (A rule occurrence in ρ' is matched to another occurrence of the same rule in ρ.)
- ψ is surjective. (For every position $k \in \mathrm{dom}(\rho)$ there is at least one $\rho' \in R$ and a position $i \in \mathrm{dom}(\rho')$ such that $\psi(\rho', i) = k$, or, informally, R "covers" ρ.)
- If $i < j$, then $\psi(\rho', i) < \psi(\rho', j)$. (So $\psi(\rho', 1)\psi(\rho', 2) \cdots$ is a scattered subword of ρ.)

Example 3. Let ρ be a run of a PDM P. Below are two distributions R and S of $\rho = r_a r_b r_b r_c r_c r_c$. On the left we have $R = \{\rho'_1, \rho'_2, \rho'_3\}$, and its embedding function ψ; on the right $S = \{\sigma'_1, \sigma'_2, \sigma'_3\}$, and its function ψ'.

ψ	1 2 3
ρ'_1	1 6
ρ'_2	1 2 5
ρ'_3	1 3 4

	1	2	3	4	5	6
$\rho = $	r_a	r_b	r_b	r_c	r_c	r_c
$\rho'_1 = $	r_a					r_c
$\rho'_2 = $	r_a	r_b			r_c	
$\rho'_3 = $	r_a		r_b	r_c		

ψ'	1 2 3 4
σ'_1	1 4
σ'_2	1 2 4 5
σ'_3	1 3 5 6

	1	2	3	4	5	6
$\rho = $	r_a	r_b	r_b	r_c	r_c	r_c
$\sigma'_1 = $	r_a			r_c		
$\sigma'_2 = $	r_a	r_b		r_c	r_c	
$\sigma'_3 = $	r_a		r_b		r_c	r_c

Lemma 5. (Distributing Lemma). *Let $u \in L(D)$, and let M be a multiset of words of $L(C)$ compatible with u. Let $v \in M$ and let ρ an accepting run of v in C that distributes to a multiset R of runs of C, and let M_R the corresponding multiset of words. Then $M \ominus \{v\} \oplus M_R$ is also compatible with u.*

The Boundedness Lemma. We are interested in distributing a multiset of runs of C into another multiset with, loosely speaking, "better" effective stack height.

Fix a run ρ of C and a distribution R of ρ with embedding function ψ. In Example 3, $(\rho)_{1..4}$ is distributed into $(\rho'_1)_{1..1}$, $(\rho'_2)_{1..2}$ and $(\rho'_3)_{1..3}$. Assume ρ is executed by one contributor. We can replace it by 3 contributors executing $\rho'_1, \rho'_2, \rho'_3$, without the rest of the network noticing any difference. Indeed, the three processes can execute r_a immediately after each other, which for the rest of the network is equivalent to the old contributor executing one r_a. Then we replace the execution of $(\rho)_{2..4}$ by $(\rho'_2)_2(\rho'_3)_{2..3}$.

We introduce some definitions allowing us to formally describe such facts. Given $k \in \mathrm{dom}(\rho)$, we denote by $c(\rho, k)$ the configuration reached by ρ after k steps. We naturally extend this notation to define $c(\rho, 0)$ as the initial configuration. We denote by $last_\psi(\rho', i)$ the largest position $k \in \mathrm{dom}(\rho')$ such that $\psi(\rho', k) \leq i$ (similarly if none exists, we fix $last_\psi(\rho', i) = 0$). Further, we denote by $c_\psi(\rho', k)$ the configuration reached by ρ' after k steps of ρ, that is, the configuration reached by ρ' after the execution of $last_\psi(\rho', k)$ transitions; formally, $c_\psi(\rho', k) = c(\rho', last_\psi(\rho', k))$.

Example 4. Let ρ, R, and ψ as in Example 3. Assuming that the PDM P has one single state p, stack symbols $\{\bot, \alpha\}$ such that the three rules r_a, r_b and r_c are given by $r_a \colon p\bot \to p\alpha\bot$, $r_b \colon p\alpha \to p\alpha\alpha$, and $r_c \colon p\alpha \to p$, then we have $c(\rho, 5) = p\alpha\bot$. Further, $last_\psi(\rho'_1, 5) = 1$, $last_\psi(\rho'_2, 5) = 3$, and $last_\psi(\rho'_3, 5) = 3$. Finally, $c_\psi(\rho'_1, 5) = p\alpha\bot$, $c_\psi(\rho'_2, 5) = p\alpha\bot$, and $c_\psi(\rho'_3, 5) = p\alpha\bot$.

Given $Z \in \mathrm{dom}(\rho)$ and $K \in \mathbb{N}$, we say that a distribution R of ρ is (Z, K)-bounded if for every $\rho' \in R$ and for every $i \leq Z$, the effective stack height of

$c_\psi(\rho', i)$ is bounded by K. Further, we say that R is synchronized if for every configuration $c(\rho, i)$ with effective stack height 1 and for every $\rho' \in R$, $c_\psi(\rho', i) = c(\rho, i)$ (same control state and same stack content), and also has effective stack height 1.[3] The Boundedness Lemma states that there is a constant N, depending only on C, such that for every run ρ of C and for every $Z \in \mathrm{dom}(\rho)$ there is a (Z, N)-bounded and synchronized distribution R_Z of ρ. The key of the proof is the following lemma.

Lemma 6. *Let $N = 2|Q_C|^2|\Gamma_C| + 1$. Let ρ be a run of C and $Z \in dom(\rho)$ be the first position of ρ such that $c(\rho, Z)$ is not N-bounded. Then there is a (Z, N)-bounded and synchronized distribution of ρ.*

Proof sketch. We construct a (Z, N)-bounded and synchronized distribution $\{\rho_a, \rho_b\}$ of ρ. Let $\alpha_{N+1}\alpha_N \cdots \alpha_1 w_0$ be the stack content of $c(\rho, Z)$. Define $\{\overrightarrow{p}_1, \overleftarrow{p}_1, \overrightarrow{p}_2, \overleftarrow{p}_2, \ldots, \overrightarrow{p}_N, \overleftarrow{p}_N\} \subseteq \mathrm{dom}(\rho)$ such that for each i, $1 \leq i \leq N$ we have $c(\rho, \overrightarrow{p}_i)$ and $c(\rho, \overleftarrow{p}_i)$ are the configurations immediately after the symbol α_i in $c(\rho, Z)$ is pushed, respectively popped and such that the stack content of each configuration between $\overrightarrow{p}_i$ (included) and $\overleftarrow{p}_i$ (excluded) equals $w_p\alpha_i\alpha_{i-1}\cdots\alpha_1 w_0$ for some $w_p \in \Gamma_C^*$. We get $c(\rho, \overrightarrow{p}_i) = q_i\alpha_i\alpha_{i-1}\ldots\alpha_0 w_0$ and $c(\rho, \overleftarrow{p}_i) = q_i'\alpha_{i-1}\ldots\alpha_0 w_0$ for some $q_i, q_i' \in Q_C$. Observe that the following holds: $\overrightarrow{p}_1 < \cdots < \overrightarrow{p}_{N-1} < \overrightarrow{p}_N < Z < \overleftarrow{p}_N < \overleftarrow{p}_{N-1} < \cdots < \overleftarrow{p}_1$.

Since $N = 2|Q_C|^2|\Gamma_C| + 1$, by the pigeonhole principle we find q, α, q' and three indices $1 \leq j_1 < j_2 < j_3 \leq N$ such that by letting $w_1 = \alpha_{j_1-1}\cdots\alpha_1$, $w_2 = \alpha_{j_2-1}\cdots\alpha_{j_1}$ and $w_3 = \alpha_{j_3-1}\cdots\alpha_{j_2}$, we have:

$$\begin{aligned}\rho =&(\rho)_{1..\overrightarrow{p}_{j_1}}\ [q\alpha w_1]\ (\rho)_{\overrightarrow{p}_{j_1+1}..\overrightarrow{p}_{j_2}}\ [q\alpha w_2 w_1]\ (\rho)_{\overrightarrow{p}_{j_2+1}..\overrightarrow{p}_{j_3}}\ [q\alpha w_3 w_2 w_1]\\ &(\rho)_{\overrightarrow{p}_{j_3+1}..\overleftarrow{p}_{j_3}}\ [q' w_3 w_2 w_1]\ (\rho)_{\overleftarrow{p}_{j_3+1}..\overleftarrow{p}_{j_2}}\ [q' w_2 w_1]\ (\rho)_{\overleftarrow{p}_{j_2+1}..\overleftarrow{p}_{j_1}}\ [q' w_1]\ (\rho)_{\overleftarrow{p}_{j_1+1}..\infty}\ .\end{aligned}$$

Here, the notation indicates that we reach configuration $[q\alpha w_1]$ after $(\rho)_{1..\overrightarrow{p}_{j_1}}$, the configuration $[q\alpha w_2 w_1]$ after $(\rho_{1..\overrightarrow{p}_{j_2}}$, etc.

Now define ρ_a from ρ by simultaneously deleting $(\rho)_{\overrightarrow{p}_{j_1+1}..\overrightarrow{p}_{j_2}}$ and $(\rho)_{\overleftarrow{p}_{j_2+1}..\overleftarrow{p}_{j_1}}$. We similarly define ρ_b by deleting $(\rho)_{\overrightarrow{p}_{j_2+1}..\overrightarrow{p}_{j_3}}$ and $(\rho)_{\overleftarrow{p}_{j_3+1}..\overleftarrow{p}_{j_2}}$.

The following shows that ρ_a defines a legal run since it is given by

$$\begin{aligned}&(\rho)_{1..\overrightarrow{p}_{j_1}}\ [q\alpha w_1]\ (\rho)_{\overrightarrow{p}_{j_2+1}..\overrightarrow{p}_{j_3}}\ [q\alpha w_3 w_1](\rho)_{\overrightarrow{p}_{j_3+1}..\overleftarrow{p}_{j_3}}\ [q' w_3 w_1]\ (\rho)_{\overleftarrow{p}_{j_3+1}..\overleftarrow{p}_{j_2}}\ [q' w_1]\\ &\qquad\qquad (\rho)_{\overleftarrow{p}_{j_1+1}..\infty}\ .\end{aligned}$$

A similar reasoning holds for ρ_b. Finally, one can show that $\{\rho_a, \rho_b\}$ is a (Z, N)-bounded and synchronized distribution of ρ.

Lemma 7. (Boundedness Lemma). *Let $N = 2|Q_C|^2|\Gamma_C| + 1$, and let ρ be a run of C. For every $Z \in dom(\rho)$ there is an (Z, N)-bounded and synchronized distribution R_Z of ρ.*

[3] Notice that the effective stack height of a configuration depends on the run it belongs to, and so $c(\rho, i) = c_\psi(\rho', i)$ does not necessarily imply that they have the same effective stack height.

The proof is by induction on Z. The distribution ψ_{Z+1}, R_{Z+1} is obtained from ψ_Z, R_Z by distributing each run ρ' of R_Z to a $(\psi_Z(\rho', Z) + 1, N)$-bounded run (applying Lemma 6).

Proof Sketch of Theorem 4. Given a run σ of D compatible with a finite multiset M of runs of C, we construct another run τ of D, and a multiset R of N-bounded runs of C_N such that τ and R are compatible as well. We consider only the special case in which M has one single element ρ (and one single copy of it). Since σ is compatible with ρ, we fix a witness $\pi \in \mathcal{S}$ such that $\pi \in \sigma \between \rho$. We construct a "lasso run" out of π of the form $\lambda_1[\lambda_2]^\omega$. It suffices to find two positions in π where the content of the store is the same, the corresponding configurations of the leader are the same, and similarly for each contributor; the fragment between these two positions can be repeated (is "pumpable").

Given a position i of π, let i_ρ and i_σ denote the corresponding positions in ρ and σ.[4] Further, for every Z let R_Z be a (Z, N)-bounded and synchronized distribution of ρ with embedding function ψ (which exists by the Boundedness Lemma). Let $R_Z(i_\rho) = \{c_\psi(\eta, i_\rho) \mid \eta \in R_Z\}$ denote the multiset of configurations reached by the runs of R_Z after i steps of π. Using Proposition 1 and that (i)the store has a finite number of values, (ii) R_Z is (Z, N)-bounded, and (iii)there are only finitely many active prefixes of length at most N, we can apply the pigeonhole principle to find a sufficiently large number Z and three positions $i < j < k \leq Z$ in π satisfying the following properties:

(1) The contents of the store at positions i and k of π coincide.
(2) The configurations $c(\sigma, i_\sigma)$ and $c(\sigma, k_\sigma)$ of the leader have effective stack height 1, same topmost stack symbol and same control state. Further, σ enters and leaves some accepting state between i_σ and k_σ.
(3) The configuration $c(\rho, j_\rho)$ has effective stack height 1.
(4) For every configuration of $R_Z(i_\rho)$ there is a configuration of $R_Z(k_\rho)$ with the same control state and active prefix, and vice versa.

Condition (4) means that, after removing the dark suffixes, $R_Z(i_\rho)$ and $R_Z(k_\rho)$ contain the same pruned configurations, although possibly a different number of times (same set, different multisets). If we obtain the same multiset, then the fragment of π between positions i and k is pumpable by (1) and (2), and we are done. Otherwise, we use (3) and the fact that R_Z is synchronized (which had not been used so far) to obtain a new distribution in which the multisets coincide. This is achieved by adding new runs to R_Z.

References

1. Abdulla, P.A., Bertrand, N., Rabinovich, A., Schnoebelen, P.: Verification of probabilistic systems with faulty communication. Inf. Comput. **202**(2), 105–228 (2005)
2. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems. In: LICS'1996. pp. 313–321. IEEE Computer Society (1996)

[4] Position p in π defines position p_σ in σ such that $(\sigma)_{1..p_\sigma} = Proj_{\Sigma_\mathcal{D}}((\pi)_{1..p})$, similarly p_ρ is defined as satisfying $(\rho)_{1..p_\rho} = Proj_{\Sigma_\mathcal{C}}((\pi)_{1..p})$.

3. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Inf. Comput. **127**(2), 91–101 (1996)
4. Aminof, B., Kotek, T., Rubin, S., Spegni, F., Veith, H.: Parameterized model checking of rendezvous systems. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 109–124. Springer, Heidelberg (2014)
5. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007)
6. Apt, K.R., Kozen, D.C.: Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett. **22**(6), 307–309 (1986)
7. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: application to model-checking. In: CONCUR'1997: Proceedings of 8th International Conference on Concurrency Theory. LNCS, vol. 1243, pp. 135–150. Springer (1997)
8. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS'1999. pp. 352–359. IEEE Computer Society (1999)
9. Esparza, J., Ganty, P., Majumdar, R.: Parameterized verification of asynchronous shared-memory systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 124–140. Springer, Heidelberg (2013)
10. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3), 675–735 (1992)
11. Grädel, E.: Subclasses of presburger arithmetic and the polynomial-time hierarchy. Theor. Comput. Sci. **56**, 289–301 (1988)
12. Hague, M.: Parameterised pushdown systems with non-atomic writes. In: Proceedings of FSTTCS'2011. LIPIcs, vol. 13, pp. 457–468. Schloss Dagstuhl (2011)
13. Meyer, R.: On boundedness in depth in the pi-calculus. In: Procedings of IFIP TCS 2008. IFIP, vol. 273, pp. 477–489. Springer (2008)
14. Pnueli, A., Xu, J., Zuck, L.D.: Liveness with $(0, 1, \infty)$-counter abstraction. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 107–122. Springer, Heidelberg (2002)
15. Verma, K.N., Seidl, H., Schwentick, T.: On the complexity of equational horn clauses. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 337–352. Springer, Heidelberg (2005)