Parameterized Communicating Automata: Complementation and Model Checking*

Benedikt Bollig[1], Paul Gastin[1], and Akshay Kumar[2]

1 **LSV, ENS Cachan & CNRS, France**
{bollig,gastin}@lsv.ens-cachan.fr
2 **Indian Institute of Technology Kanpur, India**
kakshay@iitk.ac.in

Abstract

We study the language-theoretical aspects of parameterized communicating automata (PCAs), in which processes communicate via rendez-vous. A given PCA can be run on any topology of bounded degree such as pipelines, rings, ranked trees, and grids. We show that, under a context bound, which restricts the local behavior of each process, PCAs are effectively complementable. Complementability is considered a key aspect of robust automata models and can, in particular, be exploited for verification. In this paper, we use it to obtain a characterization of context-bounded PCAs in terms of monadic second-order (MSO) logic. As the emptiness problem for context-bounded PCAs is decidable for the classes of pipelines, rings, and trees, their model-checking problem wrt. MSO properties also becomes decidable. While previous work on model checking parameterized systems typically uses temporal logics without next operator, our MSO logic allows one to express several natural next modalities.

1998 ACM Subject Classification F.1.1 [Computation by Abstract Devices]: Models of Computation, F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs

Keywords and phrases parameterized verification, complementation, monadic second-order logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.625

1 Introduction

The "regularity" of an automata model is intrinsically tied to characterizations in algebraic or logical formalisms, and to related properties such as closure under complementation and decidability of the emptiness problem. Most notably, the robustness of finite automata is witnessed by the Büchi-Elgot-Trakhtenbrot theorem, stating their expressive equivalence to monadic second-order (MSO) logic. In the past few years, this fundamental result has been extended to models of concurrent systems such as communicating finite-state machines (see [10] for an overview) and multi-pushdown automata (e. g., [11, 12]). Hereby, the system topology, which provides a set of processes and links between them, is usually supposed to be static and fixed in advance. However, in areas such as mobile computing or ad-hoc networks, it is more appropriate to design a program, and guarantee its correctness, independently of the underlying topology, so that the latter becomes a parameter of the system.

There has been a large body of literature on parameterized concurrent systems [9, 8, 6, 2, 1], with a focus on verification: Does the given system satisfy a specification independently of

* Supported by LIA InForMel.

the number of processes? A variety of different models have been introduced, covering a wide range of communication paradigms such as broadcasting, rendez-vous, token-passing, etc. So far, however, it is fair to say that there is no such thing as a canonical or "robust" model of parameterized concurrent systems.

This paper tries to take a step forward towards such a model. It is in line with a study of a *language theory* of parameterized concurrent systems that has been initiated in [3, 4]. We resume the model of parameterized communicating automata (PCAs), a conservative extension of classical communicating finite-state machines [5]. While the latter run a fixed set of processes, a PCA can be run on *any* topology of bounded degree, such as pipelines, rings, ranked trees, or grids. A topology is a graph, whose nodes represent processes that are connected via interfaces. Every process will run a local automaton executing send and receive actions, which allows it to communicate with an adjacent process in a rendez-vous fashion. As we are interested in language-theoretical properties, we associate, with a given PCA, the set of all possible executions. An execution includes the underlying topology, the events that each process executes, and the causal dependencies that exist between events. This language-theoretic view is different from most previous approaches to parameterized concurrent systems, which rather consider the transition system of reachable configurations. Yet, it will finally allow us to study such important concepts like complementation and MSO logic. Note that logical characterizations of PCAs have been obtained in [3]. However, those logics use negation in a restricted way, since PCAs are in general not complementable. This asks for restrictions of PCAs that give rise to a *robust* automata model. In this paper, we will therefore impose a bound on the number of *contexts* that each process traverses. We explain this notion below.

The efficiency of distributed algorithms and protocols is usually measured in terms of two parameters: the number n of processes, and the number k of contexts. Here, a context, sometimes referred to as *round*, restricts communication of a process to patterns such as "send a message to each neighbor and receive a message from each neighbor". In this paper, we consider more relaxed definitions where, in every context, a process may perform an unbounded number of actions. In an *interface*-context, a process can send and receive an arbitrary number of messages to/from a *fixed* neighbor. A second context-type definition allows for arbitrarily many sends to all neighbors, or receptions from a fixed neighbor.

In general, basic questions such as reachability are undecidable for PCAs, even when we restrict to simple classes of topologies such as pipelines. To get decidability, it is therefore natural to bound one of the aforementioned parameters, n or k. Bounding the number n of processes is known as *cut-off*. However, the trade-off between n and k is often in favor of an up to exponentially smaller k. Moreover, many distributed protocols actually restrict to a bounded number of contexts, such as P2P protocols and certain leader-election protocols. Therefore, bounding the parameter k seems to be an appropriate way to overcome the theoretical limitations of formally verifying parameterized concurrent systems.

The most basic verification question of context-bounded PCAs has been considered in [4]: Is there a topology that allows for an accepting run of the given PCA? In the present paper, we go beyond such nonemptiness/reachability issues and consider PCAs as language acceptors. We will show that, under suitable context bounds, PCAs form a robust automata model that is closed under complementation. Complementability relies on a disambiguation construction, which is the key technical contribution of the paper.

Our complementation result has wider applications and implications. In particular, we obtain a characterization of context-bounded PCAs in terms of MSO logic. Together with the results from [4], this implies that context-bounded model checking of PCAs against MSO

logic is decidable for the classes of pipelines, rings, and trees. Note that MSO logic is quite powerful and, unlike in [7, 2], we are not constrained to drop any (next) modality. Actually, a variety of natural next modalities can be expressed in MSO logic, such as process successor, message successor, next event on a neighboring process, etc.

Context-bounds were originally introduced for (sequential) multi-pushdown automata as models of multi-threaded recursive programs [15]. Interestingly, determinization procedures have been used to obtain complementability and MSO characterizations for context-bounded multi-pushdown automata [11, 12]. A pattern that we share with these approaches is that of computing *summaries* in a deterministic way. Overall, however, we have to use quite different techniques, which is due to the fact that, in our model, processes evolve asynchronously.

In Section 2, we settle some basic notions such as topologies and message sequence charts, which describe the behavior of a system. PCAs and their restrictions are introduced in Section 3. Section 4 presents our main technical contribution: We show that context-bounded PCAs are complementable. This result is exploited in Section 5 to obtain a logical characterization of PCAs and decidability of the model-checking problem wrt. MSO logic. Omitted proofs can be found in the full version of the paper, which is available at: `http://hal.archives-ouvertes.fr/hal-01030765/`.

2 Preliminaries

For $n \in \mathbb{N}$, we set $[n] := \{1, \dots, n\}$. Let $\mathbb{A}$ be an alphabet and I be an index set. Given a tuple $\bar{a} = (a_i)_{i \in I} \in \mathbb{A}^I$ and $i \in I$, we write $\bar{a}_i$ to denote a_i.

Topologies. We will model concurrent systems without any assumption on the number of processes. However, we will have in mind that processes are arranged in a certain way, for example as pipelines or rings. Once such a class and the number of processes are fixed, we obtain a topology. Formally, a topology is a graph. Its nodes represent processes, which are connected via interfaces. Let $\mathcal{N} = \{a, b, c, \dots\}$ be a *fixed* nonempty finite set of *interface names* (or, simply, *interfaces*). When we consider pipelines or rings, then $\mathcal{N} = \{a, b\}$ where a refers to the right neighbor and b to the left neighbor of a process, respectively. For grids, we will need two more names, which refer to adjacent processes above and below. Ranked trees require an interface for each of the (boundedly many) children of a process, as well as a pointer to the father process. As $\mathcal{N}$ is fixed, topologies are structures of bounded degree.

▶ **Definition 1.** A *topology* over $\mathcal{N}$ is a pair $\mathcal{T} = (P, \longmapsto)$ where P is the nonempty finite set of *processes* and $\longmapsto \subseteq P \times \mathcal{N} \times \mathcal{N} \times P$ is the *edge relation*. We write $p \overset{a\ b}{\longmapsto} q$ for $(p, a, b, q) \in \longmapsto$, which signifies that the a-interface of p points to q, and the b-interface of q points to p. We require that, whenever $p \overset{a\ b}{\longmapsto} q$, the following hold:

(a) $p \neq q$ (there are no self loops),

(b) $q \overset{b\ a}{\longmapsto} p$ (adjacent processes are mutually connected), and

(c) for all $a', b' \in \mathcal{N}$ and $q' \in P$ such that $p \overset{a'\ b'}{\longmapsto} q'$, we have $a = a'$ iff $q = q'$ (an interface points to at most one process, and two distinct interfaces point to distinct processes).

We do not distinguish isomorphic topologies.

▶ **Example 2.** Example topologies are depicted in Figures 1 and 2. In Figure 2, five processes are arranged as a *ring*. Formally, a ring is a topology over $\mathcal{N} = \{a, b\}$ of the form $(\{1, \dots, n\}, \longmapsto)$ where $n \geq 3$ and $\longmapsto = \{\, (i, a, b, (i \bmod n) + 1) \mid i \in [n] \} \cup \{((i \bmod n) + 1, b, a, i) \mid i \in [n]\}$. A ring is uniquely given by its number of processes. Moreover, as we do not distinguish isomorphic topologies, it does not have an "initial" process. A *pipeline* is of

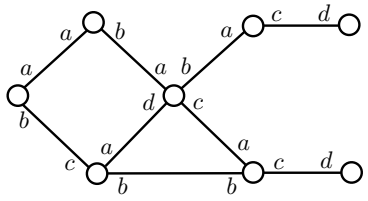

Figure 1 A topology over $\{a, b, c, d\}$.

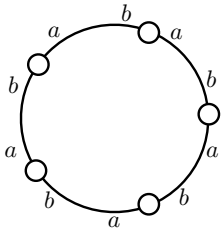

Figure 2 A ring topology.

the form $(\{1, \ldots, n\}, \longmapsto)$ where $n \geq 2$ and $\longmapsto = \{\, (i, a, b, i{+}1) \mid i \in [n{-}1]\} \cup \{(i{+}1, b, a, i) \mid i \in [n-1]\}$. Similarly, one can define ranked trees and grids [3]. ◀

MSO Logic over Topologies. The acceptance condition of a parameterized communicating automaton (PCA, as introduced in the next section) will be given in terms of a formula from *monadic second-order (MSO) logic*, which scans the final configuration reached by a PCA: the underlying topology together with the local final states in which the processes terminate. If S is the finite set of such local states, the formula thus defines a set of S-labeled topologies, i.e., structures $(P, \longmapsto, \lambda)$ where $(P, \longmapsto)$ is a topology and $\lambda : P \to S$. The logic $\mathrm{MSO}_t(S)$ is given by the grammar $\mathcal{F} ::= u \overset{a\ b}{\longmapsto} v \mid u = v \mid \lambda(u) = s \mid u \in U \mid \exists u.\mathcal{F} \mid \exists U.\mathcal{F} \mid \neg\mathcal{F} \mid \mathcal{F} \vee \mathcal{F}$ where $a, b \in \mathcal{N}$, $s \in S$, u and v are first-order variables (interpreted as processes), and U is a second-order variable (ranging over sets of processes). Note that we assume an infinite supply of variables. Given a sentence $\mathcal{F} \in \mathrm{MSO}_t(S)$ (i.e., a formula without free variables), we write $L(\mathcal{F})$ for the set of S-labeled topologies $(P, \longmapsto, \lambda)$ that satisfy $\mathcal{F}$. Hereby, satisfaction is defined in the expected manner (cf. also Section 5, presenting an extended logic).

Message Sequence Charts. Recall that our primary concern is a language-theoretic view of parameterized concurrent systems. To this aim, we associate with a system its language, i.e., the set of those behaviors that are generated by an accepting run. One single behavior is given by a message sequence chart (MSC). An MSC consists of a topology (over the given set of interfaces) and a set of events, which represent the communication actions executed by a system. Events are located on the processes and connected by process and message edges, which reflect causal dependencies (as we consider rendez-vous communication, a message edge has to be interpreted as "simultaneously").

▶ **Definition 3.** A *message sequence chart (MSC)* over $\mathcal{N}$ is a tuple $M = (P, \longmapsto, E, \lhd, \pi)$ where $(P, \longmapsto)$ is a topology over $\mathcal{N}$, E is the nonempty finite set of *events*, $\lhd \subseteq E \times E$ is the *acyclic* edge relation, which is partitioned into $\lhd_{\mathsf{proc}}$ and $\lhd_{\mathsf{msg}}$, and $\pi : E \to P$ determines the location of an event in the topology; for $p \in P$, we let $E_p := \{e \in E \mid \pi(e) = p\}$. We require that the following hold:

- $\lhd_{\mathsf{proc}}$ is a union $\bigcup_{p \in P} \lhd_p$ where each $\lhd_p \subseteq E_p \times E_p$ is the direct-successor relation of some total order on E_p,
- there is a partition $E = E_! \uplus E_?$ such that $\lhd_{\mathsf{msg}} \subseteq E_! \times E_?$ defines a bijection from $E_!$ to $E_?$,
- for all $(e, f) \in \lhd_{\mathsf{msg}}$, we have $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$ for some $a, b \in \mathcal{N}$, and
- in the graph $(E, \lhd \cup \lhd_{\mathsf{msg}}^{-1})$, there is no cycle that uses at least one $\lhd_{\mathsf{proc}}$-edge (this ensures rendez-vous communication).

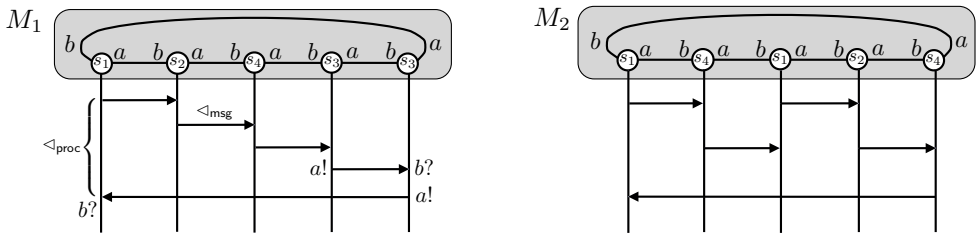

Figure 3 Two MSCs over a ring topology; local states labeling the topology in a PCA run.

Let $\Sigma = \{a! \mid a \in \mathcal{N}\} \cup \{a? \mid a \in \mathcal{N}\}$. We define a mapping $\ell_M : E \to \Sigma$ that associates with each event the type of action that it executes: For $(e, f) \in \lhd_{\mathsf{msg}}$ and $a, b \in \mathcal{N}$ such that $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$, we set $\ell_M(e) = a!$ and $\ell_M(f) = b?$.

The set of MSCs (over the fixed set $\mathcal{N}$) is denoted by $\mathbb{MSC}$. Like for topologies, we do not distinguish isomorphic MSCs.

▶ **Example 4.** Two example MSCs are depicted in Figure 3, both having the ring with five processes as underlying topology (for the moment, we ignore the state labels s_i of processes). The events are the endpoints of message arrows, which represent $\lhd_{\mathsf{msg}}$. Process edges are implicitly given; they connect successive events located on the same (top-down) process line. Finally, the mapping ℓ_{M_1} is illustrated on a few events. ◀

3 Parameterized Communicating Automata

In this section, we introduce our model of a communicating system that can be run on arbitrary topologies of bounded degree.

The idea is that each process of a given topology runs one and the same automaton, whose transitions are labeled with an action of the form $(a!, m)$, which emits a message m through interface a, or $(a?, m)$, which receives m from interface a.

▶ **Definition 5.** A *parameterized communicating automaton (PCA)* over $\mathcal{N}$ is a tuple $\mathcal{A} = (S, \iota, Msg, \Delta, \mathcal{F})$ where S is the finite set of *states*, $\iota \in S$ is the *initial state*, Msg is a nonempty finite set of *messages*, $\Delta \subseteq S \times (\Sigma \times Msg) \times S$ is the *transition relation*, and $\mathcal{F} \in \mathrm{MSO}_t(S)$ is a sentence, representing the acceptance condition.

Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC. A run of $\mathcal{A}$ on M will be a mapping $\rho : E \to S$ satisfying some requirements. Intuitively, $\rho(e)$ is the local state of $\pi(e)$ after executing e. To determine when ρ is a run, we define another mapping, $\rho^- : E \to S$, denoting the source states of a transition: whenever $f \lhd_{\mathsf{proc}} e$, we let $\rho^-(e) = \rho(f)$; moreover, if e is $\lhd_{\mathsf{proc}}$-minimal, we let $\rho^-(e) = \iota$. With this, we say that ρ is a run of $\mathcal{A}$ on M if, for all $(e, f) \in \lhd_{\mathsf{msg}}$, there are $a, b \in \mathcal{N}$ and a message $m \in Msg$ such that $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$, $(\rho^-(e), (a!, m), \rho(e)) \in \Delta$, and $(\rho^-(f), (b?, m), \rho(f)) \in \Delta$. To determine when ρ is accepting, we collect the last states of all processes and define a mapping $\lambda : P \to S$ as follows. Let $p \in P$. If $E_p = \emptyset$, then $\lambda(p) = \iota$; otherwise, $\lambda(p)$ is set to $\rho(e)$ where e is the unique $\lhd_{\mathsf{proc}}$-maximal event of p. Now, run ρ is *accepting* if $(P, \longmapsto, \lambda) \in L(\mathcal{F})$. The set of MSCs that allow for an accepting run is denoted by $L(\mathcal{A})$.

While a run of a PCA is purely operational, it is actually natural to define the acceptance condition in terms of $\mathrm{MSO}_t(S)$, which allows for a global, declarative view of the final configuration. Note that, when we restrict to pipelines, rings, or ranked trees, the acceptance condition could be defined as a finite (tree, respectively) automaton over the alphabet S.

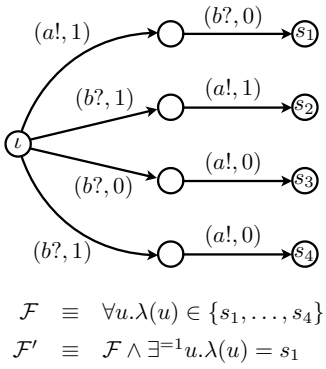

Figure 4 The PCA $\mathcal{A}'_{\text{token}}$.

▶ **Example 6.** The PCA from Figure 4 describes a simplified version of the IEEE 802.5 token-ring protocol. For illustration, we consider two different acceptance conditions, $\mathcal{F}$ and $\mathcal{F}'$, giving rise to PCAs $\mathcal{A}_{\text{token}}$ and $\mathcal{A}'_{\text{token}}$, respectively. In both cases, a single binary token, which can carry a value from $m \in \{0, 1\}$, circulates in a ring. Recall that, in a ring topology, every process has an a-neighbor and a b-neighbor (cf. Figure 2). Initially, the token has value 1. A process that has the token may emit a message and pass it along with the token to its a-neighbor. We will abstract the concrete message away and only consider the token value. Whenever a process receives the token from its b-neighbor, it will forward it to its a-neighbor, while (i) leaving the token value unchanged (the process then ends in state s_2 or s_3), or (ii) changing the token value from 1 to 0, to signal that the message has been received (the process then ends in s_4). Once the process that initially launched the token receives the token with value 0, it goes to state s_1.

Note that the acceptance condition $\mathcal{F}$ of $\mathcal{A}_{\text{token}}$ permits those configurations where all processes terminate in one of the states $s_1, \ldots, s_4$. MSC M_1 from Figure 3 depicts an execution of the protocol described above, and we have $M_1 \in L(\mathcal{A}_{\text{token}})$. The state labelings of processes indicate the final local states that are reached in an accepting run. However, one easily verifies that we also have $M_2 \in L(\mathcal{A}_{\text{token}})$, though M_2 should not be considered as an execution of a token-ring protocol: there are two processes that, independently of each other, emit a message/token and end up in s_1. To model the protocol faithfully and rule out such pathological executions, we change the acceptance condition to $\mathcal{F}'$, which adds the requirement that *exactly* one process terminates in s_1. We actually have $M_1 \in L(\mathcal{A}'_{\text{token}})$ and $M_2 \notin L(\mathcal{A}'_{\text{token}})$. ◀

Note that [3, 4] used weaker acceptance conditions, which cannot access the topology. However, Example 6 shows that an acceptance condition given as an MSO_t-formula offers some flexibility in modeling parameterized systems. For example, it can be used to simulate several process types [4], the idea being that each process runs a local automaton according to its type. All our results go through in this extended setting. Also note that messages (such as the token value in Example 6) could be made apparent in the MSCs. However, we will always need some "hidden" messages, which are common in communicating automata with fixed topology [10] and significantly extend their expressive power.

Context-Bounded PCAs. Our main results will rely on a restricted version of PCAs, where every process is constrained to execute a bounded number of *contexts*. As discussed in the introduction, contexts come very naturally when modeling distributed protocols. Actually,

the behavior of a single process is often divided into a small, or even bounded, number of *rounds*, each describing some restricted communication pattern. Usually, one considers that a round consists of sending a message to each neighbor followed by receiving a message from each neighbor [13]. In this paper, we consider *contexts*, which are somewhat more general than rounds: in a context, one may potentially execute an unbounded number of actions. Moreover, a round can be simulated by a bounded number of contexts. Actually, there exist several natural definitions. A word $w \in \Sigma^*$ is called an

- $(\mathsf{s}\oplus\mathsf{r})$-context if $w \in \{a! \mid a \in \mathcal{N}\}^*$ or $w \in \{a? \mid a \in \mathcal{N}\}^*$,
- $(\mathsf{s1}+\mathsf{r1})$-context if $w \in \{a!, b?\}^*$ for some $a, b \in \mathcal{N}$,
- $(\mathsf{s}\oplus\mathsf{r1})$-context if $w \in \{a! \mid a \in \mathcal{N}\}^*$ or $w \in \{b?\}^*$ for some $b \in \mathcal{N}$,
- intf-context if $w \in \{a!, a?\}^*$ for some $a \in \mathcal{N}$.

The context type $\mathsf{s1}\oplus\mathsf{r}$ ($w \in \{a!\}^*$ for some $a \in \mathcal{N}$ or $w \in \{b? \mid b \in \mathcal{N}\}^*$) is dual to $\mathsf{s}\oplus\mathsf{r1}$, and we only consider the latter case. All results for $\mathsf{s}\oplus\mathsf{r1}$ in this paper easily transfer to $\mathsf{s1}\oplus\mathsf{r}$.

Let $k \geq 1$ be a natural number and $ct \in \{\mathsf{s}\oplus\mathsf{r}, \mathsf{s1}+\mathsf{r1}, \mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$ be a context type. We say that $w \in \Sigma^*$ is (k, ct)-*bounded* if there are $w_1, \ldots, w_k \in \Sigma^*$ such that $w = w_1 \cdots w_k$ and w_i is a ct-context, for all $i \in [k]$. To lift this definition to MSCs $M = (P, \longmapsto, E, \lhd, \pi)$, we define the projection $M|_p \in \Sigma^*$ of M to a process $p \in P$. Let $e_1 \lhd_{\mathsf{proc}} e_2 \lhd_{\mathsf{proc}} \cdots \lhd_{\mathsf{proc}} e_n$ be the unique process-order preserving enumeration of all events of E_p. We let $M|_p = \ell_M(e_1)\ell_M(e_2)\ldots\ell_M(e_n)$. In particular, $E_p = \emptyset$ implies $M|_p = \varepsilon$. Now, we say that M is (k, ct)-*bounded* if $M|_p$ is (k, ct)-bounded, for all $p \in P$. Let $\mathbb{MSC}_{(k,ct)}$ denote the set of all (k, ct)-bounded MSCs. Given two sets L and L' of MSCs, we write $L \equiv_{(k,ct)} L'$ if $L \cap \mathbb{MSC}_{(k,ct)} = L' \cap \mathbb{MSC}_{(k,ct)}$.

▶ **Example 7.** Consider the PCAs $\mathcal{A}_{\text{token}}$ and $\mathcal{A}'_{\text{token}}$ from Figure 4. Every process executes at most two events so that we have $L(\mathcal{A}'_{\text{token}}) \subseteq L(\mathcal{A}_{\text{token}}) \subseteq \mathbb{MSC}_{(2,ct)}$ for all context types $ct \in \{\mathsf{s}\oplus\mathsf{r}, \mathsf{s1}+\mathsf{r1}, \mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$. In particular, the MSCs M_1 and M_2 from Figure 3 are $(2, ct)$-bounded.

4 Context-Bounded PCAs are Complementable

Let $ct \in \{\mathsf{s}\oplus\mathsf{r}, \mathsf{s1}+\mathsf{r1}, \mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$. We say that PCAs are *ct-complementable* if, for every PCA $\mathcal{A}$ and $k \geq 1$, we can effectively construct a PCA $\mathcal{A}'$ such that $L(\mathcal{A}') \equiv_{(k,ct)} \mathbb{MSC} \setminus L(\mathcal{A})$. In general, PCAs are not complementable, and this even holds under certain context bounds.

▶ **Theorem 8.** *Suppose $\mathcal{N} = \{a, b\}$. For all context types $ct \in \{\mathsf{s}\oplus\mathsf{r}, \mathsf{s1}+\mathsf{r1}\}$, PCAs are* not *ct-complementable.*

The proof uses results from [16, 14]. However, the situation changes when we move to context types $\mathsf{s}\oplus\mathsf{r1}$ and intf. We now present the main result of our paper:

▶ **Theorem 9.** *For all $ct \in \{\mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$, PCAs are ct-complementable.*

The theorem follows directly from a disambiguation construction, which we present as Theorem 10. We call a PCA $\mathcal{A}$ *unambiguous* if, for every MSC M, there is exactly one run (accepting or not) of $\mathcal{A}$ on M. An unambiguous PCA can be easily complemented by negating the acceptance condition.

▶ **Theorem 10.** *Given a PCA $\mathcal{A}$, a natural number $k \geq 1$, and $ct \in \{\mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$, we can effectively construct an unambiguous PCA $\mathcal{A}'$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L(\mathcal{A}')$.*

The PCA from Figure 4 is not unambiguous, since there are runs of $\mathcal{A}_{\text{token}}$ (or $\mathcal{A}'_{\text{token}}$) on the MSC M_1 from Figure 3 ending, for example, in configurations $s_1s_2s_4s_3s_3$ or $s_1s_2s_2s_4s_3$. Unfortunately, a simple power-set construction is not applicable to PCAs, due to the hidden message contents. Note that, in the fixed-topology setting, there is a commonly accepted notion of *deterministic* communicating automata [10], which is different from *unambiguous*. We do not know if Theorem 10 holds for deterministic PCAs.

Proof of Theorem 10

In the remainder of this section, we prove Theorem 10. The proof outline is as follows: We first define an intermediate model of *complete deterministic asynchronous automata (CDAAs)*. We will then show that any context-bounded PCA can be converted into a CDAA (Lemma 13) which, in turn, can be converted into an unambiguous PCA (Lemma 12).

▶ **Definition 11.** A *complete deterministic asynchronous automaton (CDAA)* over the set $\mathcal{N}$ is a tuple $\mathcal{B} = (S, \iota, (\delta_{(a,b)})_{(a,b)\in\mathcal{N}\times\mathcal{N}}, \mathcal{F})$ where S, ι, and $\mathcal{F}$ are like in PCAs and, for each $(a,b) \in \mathcal{N}\times\mathcal{N}$, we have a (total) function $\delta_{(a,b)} : (S\times S) \to (S\times S)$.

The main motivation behind introducing CDAAs is that, for a given process p, the functions $(\delta_{(a,b)})_{(a,b)\in\mathcal{N}\times\mathcal{N}}$ can effectively encode the transitions at each of the neighbors of p. Similarly to PCAs, a run of $\mathcal{B}$ on an MSC $M = (P, \longmapsto, E, \lhd, \pi)$ is a mapping $\rho : E \to S$ such that, for all $(e,f) \in \lhd_{\mathsf{msg}}$, there are $a, b \in \mathcal{N}$ satisfying $\pi(e) \overset{a\ b}{\longmapsto} \pi(f)$ and $\delta_{(a,b)}(\rho^-(e), \rho^-(f)) = (\rho(e), \rho(f))$. Whether a run is accepting or not depends on $\mathcal{F}$ and is defined exactly like in PCAs. The set of MSCs that are accepted by $\mathcal{B}$ is denoted by $L(\mathcal{B})$.

▶ **Lemma 12.** *For every CDAA $\mathcal{B}$, there is an unambiguous PCA $\mathcal{A}$ such that $L(\mathcal{B}) = L(\mathcal{A})$.*

Proof. The idea is that the messages of a PCA "guess" the current state of the receiving process. A message can only be received if the guess is correct, so that the resulting PCA is unambiguous. Let $\mathcal{B} = (S, \iota, (\delta_{(a,b)})_{(a,b)\in\mathcal{N}\times\mathcal{N}}, \mathcal{F})$ be the given CDAA. We let $\mathcal{A} = (S, \iota, \mathit{Msg}, \Delta, \mathcal{F})$ where $\mathit{Msg} = \mathcal{N}\times\mathcal{N}\times S\times S$ and Δ contains, for every transition $\delta_{(a,b)}(s_1,s_2) = (s'_1, s'_2)$, the tuples $(s_1, a!(a,b,s_1,s_2), s'_1)$ and $(s_2, b?(a,b,s_1,s_2), s'_2)$. Note that $\mathcal{A}$ is indeed unambiguous. Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC and $\rho : E \to S$. From the run definitions, we obtain that ρ is an (accepting) run of $\mathcal{B}$ on M iff it is an (accepting, respectively) run of $\mathcal{A}$ on M. It follows that $L(\mathcal{B}) = L(\mathcal{A})$. ◀

Next, we will describe how an arbitrary context-bounded PCA can be transformed into an equivalent CDAA. This construction is our key technical contribution.

▶ **Lemma 13.** *Let $ct \in \{\mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$. For every PCA $\mathcal{A}$ and $k \geq 1$, we can effectively construct a CDAA $\mathcal{B}$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L(\mathcal{B})$.*

The remainder of this section is dedicated to the proof of Lemma 13. We do the proof for the more involved case $ct = \mathsf{s}\oplus\mathsf{r1}$. Let $\mathcal{A} = (S, \iota, \mathit{Msg}, \Delta, \mathcal{F})$ be a PCA and $k \geq 1$. In the following, we will construct the required CDAA $\mathcal{B} = (S', \iota', (\delta_{(a,b)})_{(a,b)\in\mathcal{N}\times\mathcal{N}}, \mathcal{F}')$.

The idea behind our construction is that the current sending process simulates the behavior of all its neighboring receiving processes, storing all possible combinations of global source and target states. In Figure 5, in the beginning, p_2 starts sending to p_3 and p_1. Hence, p_2 keeps track of the local states at p_1 and p_3 as well. This computation spans over what we call a *zone* (the gray-shaded areas in Figure 5). Whenever a sending (receiving) process changes into a receiving (sending, respectively) process, the role of keeping track of the behavior of neighboring processes gets passed on to the new sending process, which results

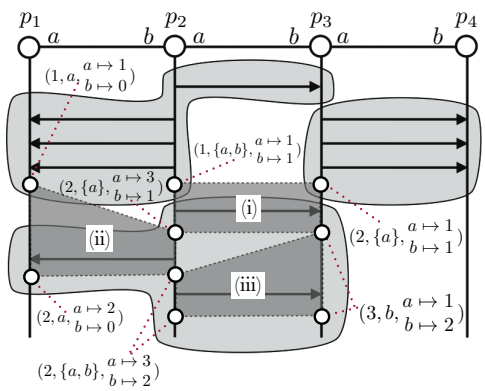

Figure 5 Computing zones in a CDAA $\mathcal{B}$.

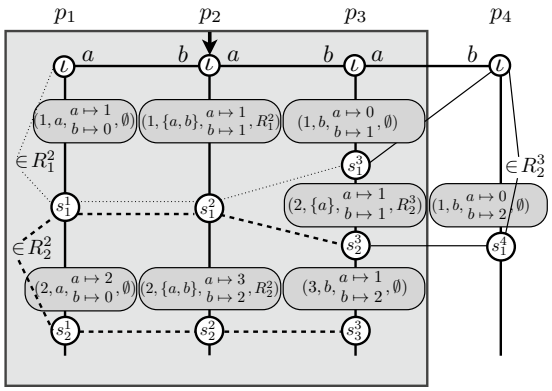

Figure 6 Illustration of $\mathcal{F}' \in \mathrm{MSO}_t(S')$.

in a zone switch. We will see that a bounded number of such changes suffice (Lemma 14). Finally, the acceptance condition $\mathcal{F}'$ checks whether the information stored at each of the processes can be coalesced to get a global run of the given PCA $\mathcal{A}$.

Zones. Let $M = (P, \longmapsto, E, \lhd, \pi)$ be an MSC. An *interval* of M is a (possibly empty) subset of E of the form $\{e_1, e_2, \ldots, e_n\}$ such that $e_1 \lhd_{\mathsf{proc}} e_2 \lhd_{\mathsf{proc}} \cdots \lhd_{\mathsf{proc}} e_n$. A *send context* of M is an interval that consists only of send events. A *receive context* of M is an interval $I \subseteq E$ such that there is $a \in \mathcal{N}$ satisfying $\ell_M(e) = a?$ for all $e \in I$.

A set $\mathcal{Z} \subseteq E$ is called a *zone* of M if there is a nonempty send context I such that the corresponding receive contexts $I_a = \{f \in E \mid e \lhd_{\mathsf{msg}} f \text{ for some } e \in I \text{ such that } \ell_M(e) = a!\}$ are intervals for all $a \in \mathcal{N}$, and $\mathcal{Z} = I \cup \bigcup_{a \in \mathcal{N}} I_a$.

Zones help us to maintain the *summary* of a possibly unbounded number of messages in a finite space. By Lemma 14 below, since there is a bound on the number of different zones for each process, the behavior of a PCA can be described succinctly by describing its action on each of the zones.

▶ **Lemma 14.** *[cf. [4]] Let $M = (P, \longmapsto, E, \lhd, \pi)$ be a $(k, \mathsf{s}{\oplus}\mathsf{r}1)$-bounded MSC. There is a partitioning of the events of M into zones such that, for each process $p \in P$, the events of E_p belong to at most $K := k \cdot (|\mathcal{N}|^2 + 2|\mathcal{N}| + 1)$ different zones.*

A CDAA that Computes Zones. We now construct a CDAA that, when running on a $(k, \mathsf{s}{\oplus}\mathsf{r}1)$-bounded MSC, computes a "greedy" zone partitioning, for which the bound K from Lemma 14 applies. We explain the intuition by means of Figure 5, which depicts an MSC along with a partitioning of events into different zones. The crucial point for processes is to recognize when to switch to a new zone. Towards this end, a *summary* of the zone is maintained. Each process stores its zone number together with the zone number of its neighboring receiving processes. A sending (receiving) process enters a new zone if the stored zone number of a neighbor does not match the actual zone number of the corresponding neighboring receiving (sending, respectively) process.

In Figure 5, the zone number of p_3 in p_2's first zone is 1. However, at the time of sending the second message from p_2 to p_3, the zone number of p_3 is 2 which does not match the information stored with p_2. This prompts p_2 to define a new zone and update the zone number of p_3.

A *sending process enters a new zone* when (a) it was a receiving process earlier, or (b) the zone number of a receiving process does not match. Similarly, a *receiving process enters a new zone* when (a) it was a sending process earlier, or (b) it was receiving previously from a different process, or (c) the zone number of the sending process does not match. This is formally defined in Equations (1) and (2) below.

We now formally describe the CDAA $\mathcal{B}$. A *zone state* is a tuple (i, τ, κ, R) where

- $i \in \{0, \dots, K\}$ is the current zone number, which indicates that a process traverses its i-th zone (or, equivalently, has switched to a new zone $i-1$ times),
- $\tau \in 2^{\mathcal{N}} \cup \mathcal{N}$ denotes the role of a process in the current zone (if $\tau \subseteq \mathcal{N}$, it has been sending through the interfaces in τ; if $\tau \in \mathcal{N}$, it is receiving from τ),
- $\kappa : \mathcal{N} \to \{0, \dots, K\}$ denotes the knowledge about each neighbor, and
- $R \subseteq (S^{\mathcal{N} \cup \{\mathsf{self}\}})^2$ is the set of possible global steps that the zone may induce; each step involves a source and target state for the current process as well as its neighbors. As the sending process simulates the receivers' steps, we let $R = \emptyset$ whenever $\tau \in \mathcal{N}$.

Let Z be the set of zone states. For $(a, b) \in \mathcal{N} \times \mathcal{N}$, we define a *partial* "update" function $\delta^{\text{zone}}_{(a,b)} : (Z \times Z) \rightharpoonup (Z \times Z)$ by

$$\delta^{\text{zone}}_{(a,b)}((i_1, \tau_1, \kappa_1, R_1), (i_2, \tau_2, \kappa_2, R_2)) = ((i_1', \tau_1', \kappa_1[a \mapsto i_2'], R_1'), (i_2', b, \kappa_2[b \mapsto i_1'], \emptyset))$$

where

$$i_1' = \begin{cases} i_1 + 1 & \text{if } i_1 = 0 \text{ or } \tau_1 \in \mathcal{N} \text{ or } (a \in \tau_1 \text{ and } \kappa_1(a) \neq i_2) \\ i_1 & \text{otherwise} \end{cases} \tag{1}$$

$$i_2' = \begin{cases} i_2 + 1 & \text{if } \tau_2 \neq b \text{ or } \kappa_2(b) \neq i_1 \\ i_2 & \text{otherwise} \end{cases} \tag{2}$$

$$\tau_1' = \begin{cases} \{a\} & \text{if } i_1' = i_1 + 1 \\ \tau_1 \cup \{a\} & \text{otherwise} \end{cases} \qquad R_1' = \begin{cases} R & \text{if } i_1' = i_1 + 1 \\ R_1 \circ R & \text{otherwise} \end{cases}$$

with R being the set of pairs $(\bar{s}, \bar{s}') \in (S^{\mathcal{N} \cup \{\mathsf{self}\}})^2$ such that there is $m \in \mathit{Msg}$ with $(\bar{s}_{\mathsf{self}}, (a!, m), \bar{s}'_{\mathsf{self}}) \in \Delta$, $(\bar{s}_a, (b?, m), \bar{s}'_a) \in \Delta$, and $\bar{s}_c = \bar{s}'_c$ for all $c \in \mathcal{N} \setminus \{a\}$.

The function $\delta^{\text{zone}}_{(a,b)}$ is illustrated in Figure 5 (omitting the R-component) for the three different cases that can occur: (i) both processes increase their zone number; (ii) only the receiver increases its zone number; (iii) none of the processes increases its zone number.

A state of $\mathcal{B}$ is a sequence of zone states, so that a process can keep track of the zones that it traverses. Formally, we let S' be the set of words over Z of the form $(0, \emptyset, \kappa_0, \emptyset)(1, \tau_1, \kappa_1, R_1) \dots (n, \tau_n, \kappa_n, R_n)$ where $n \in \{0, \dots, K\}$ and $\kappa_0(a) = 0$ for all $a \in \mathcal{N}$. The initial state is $\iota' = (0, \emptyset, \kappa_0, \emptyset)$. Actually, S' will also contain a distinguished sink state, as explained below. Note that the size of S' is exponential in K (and, therefore, in k).

We are now ready to define the transition function $\delta_{(a,b)} : (S' \times S') \to (S' \times S')$. Essentially, we take $\delta^{\text{zone}}_{(a,b)}$, but we append a new zone state when the zone number is increased. Let $z_1 = (i_1, \tau_1, \kappa_1, R_1) \in Z$ and $z_2 = (i_2, \tau_2, \kappa_2, R_2) \in Z$. Moreover, suppose $\delta^{\text{zone}}_{(a,b)}(z_1, z_2) = (z_1', z_2')$ where $z_1' = (i_1', \tau_1', \kappa_1', R_1')$ and $z_2' = (i_2', \tau_2', \kappa_2', R_2')$. Then, we let

$$\delta_{(a,b)}(w_1 z_1 \,,\, w_2 z_2) = \begin{cases} (w_1 z_1' \,,\, w_2 z_2') & \text{if } i_1' = i_1 \text{ and } i_2' = i_2 \\ (w_1 z_1' \,,\, w_2 z_2 z_2') & \text{if } i_1' = i_1 \text{ and } i_2' = i_2 + 1 \\ (w_1 z_1 z_1' \,,\, w_2 z_2 z_2') & \text{if } i_1' = i_1 + 1 \text{ and } i_2' = i_2 + 1 \end{cases}$$

Note that the case $i_1' = i_1 + 1 \wedge i_2' = i_2$ can actually never happen. Nonetheless, $\delta_{(a,b)}$ is still a partial function. However, adding a sink state, we easily obtain a function that is complete.

The Acceptance Condition. It remains to determine the acceptance condition of $\mathcal{B}$. The formula $\mathcal{F}' \in \mathrm{MSO}_t(S')$ will check whether there is a concrete choice of local states that is consistent with the zone abstraction and, in particular, with the relations R collected during that run in the zone states. Let T be the set of sequences of the form $\iota s_1 \dots s_n$ where $n \in \{0, \dots, K\}$ and $s_i \in S$ for all i. The idea is that s_i is the local state that a process reaches *after* traversing its i-th zone. The formula will now guess such a sequence for every process and check if this choice matches the abstract run. To verify if the local states correspond to the relation R stored in some constituent sending process p, it is sufficient to look at the adjacent neighbors of p.

This is illustrated in Figure 6 for the zone abstraction from Figure 5. Process p_2, for example, stores both the relations R_1^2 and R_2^2, and we have to check if this corresponds to the sequences from T that the formula had guessed for every process (the white circles). To do so, it is indeed enough to look at the neighborhood of p_2, which is highlighted in gray. The guess is accepted only if the state at the beginning of a zone matches the state at the end of the previous zone. For example, in Figure 6, the formula collects the pair of tuples $((\iota, \iota, \iota), (s_1^1, s_1^2, s_1^3))$ and verifies if it is contained in R_1^2. Also, it collects the pair $((s_1^1, s_1^2, s_2^3), (s_2^1, s_2^2, s_3^3))$ and checks if it is contained in R_2^2. Similarly, looking at the neighborhood of p_3, it verifies whether $((s_1^2, s_1^3, \iota), (s_1^2, s_2^3, s_1^4)) \in R_2^3$.

Let us be more precise. Suppose the final configuration reached by $\mathcal{B}$ is $(P, \longmapsto, \lambda')$ with $\lambda' : P \to S'$. By means of second-order variables U_t, with t ranging over T, the formula $\mathcal{F}'$ guesses an assignment $\sigma : P \to T$. It will then check that, for all $p \in P$ with, say, $\lambda'(p) = \iota'(1, \tau_1, \kappa_1, R_1) \dots (n_p, \tau_{n_p}, \kappa_{n_p}, R_{n_p}) \in S'$, the following hold:

- the sequence $\sigma(p)$ is of the form $s_0 s_1 \dots s_{n_p}$ (in the following, we let $\sigma(p)_i$ refer to s_i),
- for all $i \in [n_p]$ with $\tau_i \subseteq \mathcal{N}$, there is $(\bar{s}, \bar{s}') \in R_i$ such that (i) $\bar{s}_{\mathsf{self}} = s_{i-1}$ and $\bar{s}'_{\mathsf{self}} = s_i$, and (ii) for all $p \overset{a\ b}{\longmapsto} q$ such that $a \in \tau_i$, we have $\bar{s}_a = \sigma(q)_{\kappa_i(a)-1}$ and $\bar{s}'_a = \sigma(q)_{\kappa_i(a)}$.

These requirements can be expressed in $\mathrm{MSO}_t(S')$. Finally, to incorporate the acceptance condition $\mathcal{F} \in \mathrm{MSO}_t(S)$, we simply replace an atomic formula $\lambda(u) = s$, where $s \in S$, by the disjunction of all formulas $u \in U_t$ such that $t \in T$ ends in s. This concludes the construction of the CDAA $\mathcal{B}$.

5 Monadic Second-Order Logic

MSO logic over MSCs is two-sorted, as it shall reason about processes and events. By $u, v, w, \dots$ and $U, V, W, \dots$, we denote first-order and second-order variables, which range over processes and sets of processes, respectively. Moreover, by $x, y, z, \dots$ and $X, Y, Z, \dots$, we denote variables ranging over (sets of, respectively) events. The logic MSO_m is given by the grammar $\varphi ::= u \overset{a\ b}{\longmapsto} v \mid u = v \mid u \in U \mid \exists u.\varphi \mid \exists U.\varphi \mid \neg\varphi \mid \varphi \vee \varphi \mid x \lhd_{\mathsf{proc}} y \mid x \lhd_{\mathsf{msg}} y \mid x = y \mid x@u \mid x \in X \mid \exists x.\varphi \mid \exists X.\varphi$ where $a, b \in \mathcal{N}$.

MSO_m formulas are interpreted over MSCs $M = (P, \longmapsto, E, \lhd, \pi)$. Hereby, free variables u and x are interpreted by a function $\mathcal{I}$ as a process $\mathcal{I}(u) \in P$ and an event $\mathcal{I}(x) \in E$, respectively. Similarly, U and X are interpreted as sets. We write $M, \mathcal{I} \models u \overset{a\ b}{\longmapsto} v$ if $\mathcal{I}(u) \overset{a\ b}{\longmapsto} \mathcal{I}(v)$ and $M, \mathcal{I} \models x@u$ if $\pi(\mathcal{I}(x)) = \mathcal{I}(u)$. Thus, $x@u$ says that "x is located at u". The semantics of other formulas is as expected. When φ is a sentence, i.e., a formula without free variables, then its truth value is independent of an interpretation function so that we can simply write $M \models \varphi$ instead of $M, \mathcal{I} \models \varphi$. The set of MSCs M such that $M \models \varphi$ is denoted by $L(\varphi)$.

▶ **Example 15.** Let us resume the token-ring protocol from Example 6. We would like to express that there is a process that emits a message and gets an acknowledgment that

results from a sequence of forwards through interface a. We first let $\mathrm{fwd}(x,y) \equiv x \overset{a\ b}{\longmapsto} y \wedge \exists z.(x \lhd_{\mathsf{proc}} z \lhd_{\mathsf{msg}} y)$ where $x \overset{a\ b}{\longmapsto} y$ is a shorthand for $\exists u.\exists v.(x@u \wedge y@v \wedge u \overset{a\ b}{\longmapsto} v)$. It is well known that the transitive closure of the relation induced by $\mathrm{fwd}(x,y)$ is definable in MSO_m-logic, too. Let $\mathrm{fwd}^+(x,y)$ be the corresponding formula. It expresses that there is a sequence of events leading from x to y that alternatingly takes process and message edges, hereby following the causal order. With this, the desired formula is $\varphi \equiv \exists x,y,z.(x \lhd_{\mathsf{proc}} y \wedge x \lhd_{\mathsf{msg}} z \wedge x \overset{a\ b}{\longmapsto} z \wedge \mathrm{fwd}^+(z,y)) \in \mathrm{MSO}_m$. Consider Figures 3 and 4. We have $M_1 \models \varphi$ and $M_2 \not\models \varphi$, as well as $L(\mathcal{A}'_{\mathrm{token}}) \subseteq L(\varphi)$. ◀

▶ **Theorem 16.** *Let $ct \in \{\mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$, $k \geq 1$, and $L \subseteq \mathbb{MSC}$. There is a PCA $\mathcal{A}$ such that $L(\mathcal{A}) \equiv_{(k,ct)} L$ iff there is a sentence $\varphi \in \mathrm{MSO}_m$ such that $L(\varphi) \equiv_{(k,ct)} L$.*

The direction "$\Longrightarrow$" follows a standard pattern and is actually independent of a context bound. For the direction "$\Longleftarrow$", we proceed by induction, crucially relying on Theorem 9. Note that there are some subtleties in the translation, which arise from the fact that MSO_m mixes event and process variables.

By the results from [4], we obtain decidability of MSO_m model checking as a corollary.

▶ **Theorem 17.** *Let $\mathfrak{T}$ be one of the following: the class of rings, the class of pipelines, or the class of ranked trees. The following problem is decidable, for all $ct \in \{\mathsf{s}\oplus\mathsf{r1}, \mathsf{intf}\}$:*

Input:	*A PCA $\mathcal{A}$, a sentence $\varphi \in \mathrm{MSO}_m$, and $k \geq 1$.*
Question:	*Do we have $M \models \varphi$ for all MSCs $M = (P, \longmapsto, E, \lhd, \pi) \in L(\mathcal{A}) \cap \mathbb{MSC}_{(k,ct)}$ such that $(P, \longmapsto) \in \mathfrak{T}$?*

Note that MSO_m is a powerful logic and it may actually be used for the verification of extended models that involve registers to store process identities (pids). MSO logic is able to trace back the origin of a pid so that an additional equality predicate on pids can be reduced to an MSO formula over a finite alphabet. This would allow us to model and verify leader-election protocols. It will be worthwhile to explore this in future work.

References

1. P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *VMCAI'13*, volume 7737 of *LNCS*, pages 476–495. Springer, 2013.
2. B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281. Springer, 2014.
3. B. Bollig. Logic for communicating automata with parameterized topology. In *CSL-LICS'14*. ACM Press, 2014.
4. B. Bollig, P. Gastin, and J. Schubert. Parameterized Verification of Communicating Automata under Context Bounds. In *RP'14*, volume 8762 of *LNCS*, pages 45–57, 2014.
5. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2), 1983.
6. G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *FoSSaCS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.
7. E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.
8. J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS'14*, volume 25 of *LIPIcs*, pages 1–10, 2014.
9. J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, volume 8044 of *LNCS*, pages 124–140. Springer, 2013.
10. B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fundam. Inform.*, 80(1-3):147–167, 2007.

11 S. La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. In *LATIN'10*, volume 6034 of *LNCS*, pages 96–107. Springer, 2010.

12 S. La Torre, M. Napoli, and G. Parlato. Scope-bounded pushdown languages. In *DLT'14*, volume 8633 of *LNCS*, pages 116–128. Springer, 2014.

13 N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

14 O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. *Information and Computation*, 179(2):356–383, 2002.

15 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

16 W. Thomas. Elements of an automata theory over partial orders. In *POMIV'96*, volume 29 of *DIMACS*. AMS, 1996.