An Automata-Based Approach for Synchronizable Mailbox Communication

Romain Delpy
LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Anca Muscholl
LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Grégoire Sutre
LaBRI, Univ. Bordeaux, CNRS, Bordeaux INP, Talence, France

Abstract

We revisit finite-state communicating systems with round-based communication under mailbox semantics. Mailboxes correspond to one FIFO buffer per process (instead of one buffer per pair of processes in peer-to-peer systems). Round-based communication corresponds to sequences of rounds in which processes can first send messages, then only receive (and receives must be in the same round as their sends). A system is called synchronizable if every execution can be re-scheduled into an equivalent execution that is a sequence of rounds. Previous work mostly considered the setting where rounds have fixed size. Our main contribution shows that the problem whether a mailbox communication system complies with the round-based policy, with no size limitation on rounds, is PSPACE-complete. For this we use a novel automata-based approach, that also allows to determine the precise complexity (PSPACE) of several questions considered in previous literature.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Concurrent programming, Mailbox communication, Verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.22

Related Version *Full Version*: `https://arxiv.org/abs/2407.06968` [6]

Funding This work was (partially) supported by the grant ANR-23-CE48-0005 of the French National Research Agency ANR (project PaVeDyS).

1 Introduction

Message-passing is a key synchronization feature for concurrent programming and distributed systems. In this model, processes running asynchronously synchronize by exchanging messages over unbounded channels. The usual semantics is based on peer-to-peer communication, which is very popular for reasoning about telecommunication protocols. More recently, mailbox communication received increased attention because of its usage in multi-thread programming, as provided by languages like Rust or Erlang. Mailbox communication means that every process has a single incoming communication buffer on which incoming messages from other processes are multiplexed (a mailbox).

Message-passing programs are well-known to be challenging for formal verification since they can easily simulate Turing machines with unbounded channels. Some approximation techniques can help to recover decidability. Among the best known approaches are lossy channel systems [1, 9] and partial-order methods [14]. The latter tightly relate to (high-level) message sequence charts (HMSC), a communication formalism capturing multi-party session types [18, 16, 17]. An HMSC protocol is a graph with nodes labelled by communication scenarios, a.k.a. message sequence charts. Processes still evolve asynchronously, so that the division into nodes cannot be enforced by global synchronization. Such round-based communication is actually quite frequent in distributed computing, for example as building

35th International Conference on Concurrency Theory (CONCUR 2024).
Editors: Rupak Majumdar and Alexandra Silva; Article No. 22; pp. 22:1–22:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

block in the Heard-Of model [5]. Often a distributed protocol consists of several rounds, where each round first has a phase where processes only send messages, then a phase where they only receive. We refer to such rounds as `sr`-rounds.

Recently `sr`-round-based communication and mailbox communication were considered together in [3]. It turned out that this combination is very attractive for formal verification. The paper [3] proposed a model where `sr`-rounds have fixed size, and showed that control-state reachability in this model becomes decidable (in PSPACE). The question whether a system complies with the `sr`-round model with given round size was shown to be decidable in [7]. It is also known how to decide if a system complies with the `sr`-round model when the round size is not known in advance [12]. All these properties motivate a genuine interest in the `sr`-round model on top of mailbox communication. A bit surprisingly, apart from control-state reachability, similar questions were shown to be undecidable for peer-to-peer communication [10].

In this paper we revisit the framework of [3] and propose an automata-based approach to deal with systems complying with the `sr`-round mailbox model (we refer to this property as *mb-synchronizability*). Importantly, we do not impose any size restriction on the rounds, as in previous works. This makes sense, because even when we can infer an upper bound on the size as in [12], this upper bound is exponential in the number of processes, so its practical use is somewhat limited. We establish that the complexity of all problems listed below is PSPACE-complete for *mb-synchronizable* systems:

- Global-state reachability (Theorem 3.6).
- Model-checking against a reasonable class of regular properties (Theorem 4.3).
- Check if a peer-to-peer system can be simulated as a mailbox system (modulo rescheduling executions, Theorem 4.8).

Our main result is that one can check in PSPACE if a system is `mb`-synchronizable (Theorem 5.16), the complexity being tight. An interesting byproduct of our results is that when we fix the number of processes all the problems above can be solved in PTIME (actually NLOGSPACE).

Comparison with related work. Our technique helps to establish the precise complexity of several problems considered in the papers mentioned above. To be precise, our definition of `sr`-round mailbox model (*mb-synchronizability*) slightly differs from the one used in [3, 7, 12] (but coincides with a variant introduced in [2]). The latter paper uses a partial-order variant of PDL (LCPDL) to show an EXPTIME upper bound for the synchronizability problem for their notion of synchronizability. Using MSO logic and special tree-width, the paper [2] also shows that checking if a system is synchronizable with fixed round size is decidable. Knowing if a round size exists is shown to be decidable with elementary complexity in [12], without exact bounds.

For convenience, technical terms and notations in the electronic version of this manuscript are hyper-linked to their definitions (cf. `https://ctan.org/pkg/knowledge`).

Proofs that are missing in the main text can be found in the full version of the paper [6].

2 Message-passing systems and synchronizability

Throughout the paper, $\mathbb{P}$ denotes a finite non-empty set of *processes*, and $\mathbb{M}$ denotes a finite non-empty set of *message contents*. We consider here peer-to-peer communication between distinct processes. Formally, the set of (communication) *channels* is the set *Ch* of all pairs $(p, q) \in \mathbb{P} \times \mathbb{P}$ such that $p \neq q$, and the set of (communication) *actions* is

$Act = \{p!q(m), q?p(m) \mid (p,q) \in Ch, m \in \mathbb{M}\}$. An action $p!q(m)$ denotes a *send* by p of message m to q and an action $p?q(m)$ denotes a *receive* by p of message m from q. In both cases, the process performing the action is p. Throughout the paper, we let S and R denote the sets of send actions and receive actions, formally, $S = \{p!q(m) \mid (p,q) \in Ch, m \in \mathbb{M}\}$ and $R = \{p?q(m) \mid (q,p) \in Ch, m \in \mathbb{M}\}$.

A communicating finite state machine [4] is a finite set of processes that exchange messages, each process being given as a finite LTS. Recall that a (finite) *labeled transition system*, *LTS* for short, is a quadruple $(L, A, \rightarrow, i)$ where L is a (finite) set of *states*, A is a finite alphabet, $\rightarrow \subseteq L \times A \times L$ is a set of *transitions*, and $i \in L$ is an *initial* state. We will sometimes consider LTS without initial state. In the following definition, Act_p denotes the set of actions $a \in Act$ performed by p.

▶ **Definition 2.1** (Communicating Finite-State Machine). *A* CFM *is a tuple $\mathcal{A} = (\mathcal{A}_p)_{p\in\mathbb{P}}$, where each $\mathcal{A}_p$ is a finite LTS $\mathcal{A}_p = (L_p, Act_p, \rightarrow_p, i_p)$. States in L_p are called* local states. *The* size *of $\mathcal{A}$ is defined as $\sum_{p\in\mathbb{P}}(|L_p| + |\rightarrow_p|)$.*

In this paper, we mainly study and compare two semantics of communication: peer-to-peer and mailbox. These two semantics differ in the implementation of the communication network. In the *peer-to-peer semantics*, each channel (p,q) is implemented by a dedicated fifo buffer. This is the classical semantics for communicating finite-state machines [4]. In the *mailbox semantics*, each process q is equipped with a fifo buffer that acts as a *mailbox*: all messages towards q are enqueued in this buffer. Put differently, the channels (p,q) with same receiver q are multiplexed into a single buffer.

We define both semantics of CFM jointly, by viewing channels and mailboxes as (fifo) message buffers:

▶ **Definition 2.2** (Process network). *A* process network *over $\mathbb{P}$ is a pair $\mathcal{N} = (\mathsf{B}, \mathsf{bf})$ where B is a finite set of fifo* buffers *and $\mathsf{bf} : Ch \rightarrow \mathsf{B}$ is a map that assigns a buffer to each channel.*

The peer-to-peer semantics is induced by the process network $\mathsf{p2p} = (\mathsf{B}, \mathsf{bf})$ where $\mathsf{B} = Ch$ and bf is the identity. Here, B coincides with the set of communication channels. The mailbox semantics is induced by the process network $\mathtt{mb} = (\mathsf{B}, \mathsf{bf})$ where $\mathsf{B} = \mathbb{P}$ and $\mathsf{bf}(p,q) = q$. Here, B is a set of mailboxes, one per process.

▶ Remark 2.3. For both peer-to-peer semantics and mailbox semantics we have that the buffer determines the recipient: $\mathsf{bf}(p,q) = \mathsf{bf}(p',q')$ implies $q = q'$. We call such process networks *many-to-one*.

Given a CFM and a process network we define the associated global transition system:

▶ **Definition 2.4** (Global transition system). *Let $\mathcal{A} = (\mathcal{A}_p)_{p\in\mathbb{P}}$ be a CFM, and $\mathcal{N} = (\mathsf{B}, \mathsf{bf})$ be a process network over $\mathbb{P}$. The* global transition system *associated with $\mathcal{A}, \mathcal{N}$ is the LTS $\mathcal{T}_{\mathcal{N}}(\mathcal{A}) = (C_{\mathcal{A}}, Act, \rightarrow_{\mathcal{A}}, c_{in})$ with set of configurations $C_{\mathcal{A}} = G \times ((Ch \times \mathbb{M})^*)^{\mathsf{B}}$ consisting of* global states *$G = \prod_{p\in\mathbb{P}} L_p$ (i.e., products of local states) and buffer contents, with $((\ell_p)_{p\in\mathbb{P}}, (w_{\mathsf{b}})_{\mathsf{b}\in\mathsf{B}}) \xrightarrow{a}_{\mathcal{A}} ((\ell'_p)_{p\in\mathbb{P}}, (w'_{\mathsf{b}})_{\mathsf{b}\in\mathsf{B}})$ if*

- *$\ell_p \xrightarrow{a}_p \ell'_p$ and $\ell_q = \ell'_q$ for $q \neq p$, where p is the process performing a.*
- *Send actions: if $a = p!q(m)$ then $w'_{\mathsf{b}} = w_{\mathsf{b}}\,((p,q),m)$ and $w'_{\mathsf{b}'} = w_{\mathsf{b}'}$ for $\mathsf{b}' \neq \mathsf{b}$, where $\mathsf{b} = \mathsf{bf}(p,q)$.*
- *Receive actions: if $a = p?q(m)$ then $((p,q),m)\,w'_{\mathsf{b}} = w_{\mathsf{b}}$ and $w'_{\mathsf{b}'} = w_{\mathsf{b}'}$ for $\mathsf{b}' \neq \mathsf{b}$, where $\mathsf{b} = \mathsf{bf}(p,q)$.*

The initial configuration is $c_{in} = ((i_p)_{p\in\mathbb{P}}, \varepsilon^{\mathsf{B}})$.

An *execution* of $\mathcal{T}_\mathcal{N}(\mathcal{A})$ is a sequence $\rho = c_0 \xrightarrow{a_1} c_1 \cdots \xrightarrow{a_n} c_n$ with $c_i \in C_\mathcal{A}$ such that $c_{i-1} \xrightarrow{a_i}_\mathcal{A} c_i$ for every i. The sequence $a_1 \cdots a_n$ is the *label* of the execution. The execution is *initial* if $c_0 = c_{in}$.

▶ Remark 2.5. Note that in the definition above we added the channel name to the message content inserted in a buffer. This is to exclude executions like $p!q(m)\, q?r(m)$ with $p \neq r$. Without this addition such executions would be allowed in the mailbox semantics, which is clearly not intended.

▶ **Definition 2.6** (Trace)**.** *A* trace *of a CFM $\mathcal{A}$ over a process network $\mathcal{N}$ is a sequence $u \in Act^*$ such that there exists an initial execution of $\mathcal{T}_\mathcal{N}(\mathcal{A})$ labelled by u. The set of all traces of $\mathcal{A}$ is denoted by $Tr_\mathcal{N}(\mathcal{A})$.*

As we will also need to consider infixes of executions, we introduce action sequences which are coherent w.r.t. the fifo behavior that we expect from a process network:

▶ **Definition 2.7** (Viable sequence)**.** *Let $\mathcal{N} = (\mathsf{B}, \mathsf{bf})$ be a process network. A sequence of actions $v \in Act^*$ is called* $\mathcal{N}$-viable *if for every buffer* $\mathsf{b} \in \mathsf{B}$*:*

- *for every prefix u of v, the number of receives from* b *in u is less or equal the number of sends to* b *in u;*
- *for every k, if the k-th receive from* b *in v has label $q?p(m)$ then the k-th send to* b *in v has label $p!q(m)$.*

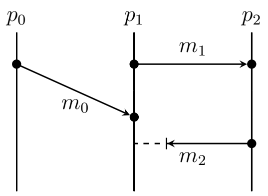
There is a strong connection between traces and viable sequences. For every sequence $u \in Act^*$, u is a trace of $\mathcal{A}$ over $\mathcal{N}$ iff u is $\mathcal{N}$-viable and u is recognized by $\prod_{p\in\mathbb{P}} \mathcal{A}_p$. Here, $\prod_{p\in\mathbb{P}} \mathcal{A}_p$ denotes the asynchronous product of the LTS $\mathcal{A}_p$, viewed as automata with every state final.

▶ Remark 2.8. It is easy to see that if a sequence is `mb`-viable then it is also `p2p`-viable. In fact, for every process network $\mathcal{N}$, we have that $\mathcal{N}$-viability implies `p2p`-viability. However, the converse is not true. For example, $p_0!p_1(m_0)\, p_2!p_1(m_1)\, p_1?p_2(m_1)$ is `p2p`-viable, but not `mb`-viable because m_1 is enqueued after m_0 in p_1's mailbox, so it cannot be received first.

The classical *happens-before* relation [15], frequently used in reasoning about distributed systems, orders the actions of each process and every (matched) send action before its matching receive. The happens-before relation naturally associates a partial order with every trace, known as *message sequence chart*:

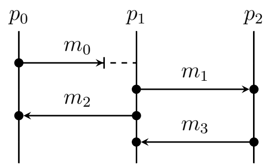
▶ **Definition 2.9** (Message Sequence Chart)**.** *An* MSC *over $\mathbb{P}$ is an Act-labeled partially ordered set $\mathcal{M} = (E, \leq_{hb}, \lambda)$ of events E, with $\lambda : E \to Act$ and $\leq_{hb} = (\leq_\mathbb{P} \cup \mathrm{msg})^*$ the least partial order containing the relations $\leq_\mathbb{P}$ and* msg*, which are defined as:*

1. *For every process p, the set of events on p is totally ordered by $\leq_\mathbb{P}$, and $\leq_\mathbb{P}$ is the union of these total orders.*
2. msg *is the set of matching send/receive event pairs. In particular, $(e, f) \in \mathrm{msg}$ implies $\lambda(e) = p!q(m)$ and $\lambda(f) = q?p(m)$ for some $p, q \in \mathbb{P}$ and $m \in \mathbb{M}$. Moreover,* msg *is a partial bijection between sends and receives such that every receive is paired with a (unique) send. A send is called* matched *if it is in the domain of* msg*, and* unmatched *otherwise.*



$p_0!p_1(m_0)\, p_1!p_2(m_1)\, p_2?p_1(m_1)\, p_2!p_1(m_2)\, p_1?p_0(m_0)$

(a) A sequence and its MSC. An unmatched send action is marked by a special arrowhead, as for m_2.



$p_0!p_1(m_0)\, p_1!p_2(m_1)\, p_2?p_1(m_1)$
$p_1!p_0(m_2)\, p_0?p_1(m_2)\, p_2!p_1(m_3)\, p_1?p_2(m_3)$

(b) A *weakly-synchronous* sequence [2] that is not mb-synchronizable.

Figure 1 Two examples of MSCs.

The fifo behavior of message buffers implies that not every MSC arises as possible behavior. We formalise this for any process network $\mathcal{N} = (\mathsf{B}, \mathsf{bf})$ by defining a *buffer order*[1] $<_{\mathcal{N}}$ on sends to the same buffer. Let $e <_{\mathcal{N}} e'$ if e, e' are of type $p!q$ and $s!r$, resp., with $\mathsf{bf}(p,q) = \mathsf{bf}(s,r)$, and

- either e is matched and e' is unmatched,
- or $(e,f), (e',f') \in \text{msg}$ and $f <_{\mathbb{P}} f'$.

▶ **Definition 2.10** (Valid MSC). *Given a process network $\mathcal{N}$, an MSC $\mathcal{M} = (E, \leq_{hb}, \lambda)$ is called $\mathcal{N}$*-valid *if the relation $(<_{hb} \cup <_{\mathcal{N}})$ is acyclic.*

It is easy to see that an MSC is `p2p`-valid iff matched messages on any channel (p,q) never overtake and unmatched sends by p to q are $\leq_{\mathbb{P}}$-ordered after the matched sends. An MSC is `mb`-valid iff for any sends $s <_{\text{hb}} s'$ to the same process, either they are both matched and their receives satisfy $r <_{\mathbb{P}} r'$, or s' is unmatched. Figure 1a shows an `mb`-valid MSC. An `mb`-valid MSC is the same as an MSC obtained from a trace that satisfies *causal delivery* in [3], and it is called *mailbox MSC* in [2].

If $u = u[1] \cdots u[n]$ is a `p2p`-viable sequence of actions then we can associate an MSC with u by setting $\mathtt{msc}(u) = (E, \leq_{\text{hb}}, \lambda)$ with $E = \{e_1, \ldots, e_n\}$, $\lambda(e_i) = u[i]$, and the orders defined as expected:

- $e_i \leq_{\mathbb{P}} e_j$ if $u[i]$ and $u[j]$ are performed by the same process and $i \leq j$.
- $(e_i, e_j) \in \text{msg}$ if there exists $k \geq 1$ and a buffer $\mathsf{b} \in \mathit{Ch}$ such that $u[i]$ is the k-th send to b and $u[j]$ is the k-th receive from b.

Note that $\mathtt{msc}(u)$ only depends (up to isomorphism) on the projection of u on each process.

Caveat. Throughout the paper we switch between reasoning on $\mathcal{N}$-viable sequences (when we use automata) and their associated MSC (when we use partial orders). So when we refer to a position in a (viable) sequence u we often see it directly as an event of $\mathtt{msc}(u)$, without further mentioning it.

▶ Remark 2.11. By definition, for any $\mathcal{N}$-viable sequence u the associated MSC $\mathtt{msc}(u)$ is $\mathcal{N}$-valid. For the converse, if the process network is many-to-one and the MSC $\mathcal{M}$ is $\mathcal{N}$-valid then every (labelled) linearization of the partial order $(<_{\text{hb}} \cup <_{\mathcal{N}})^*$ of $\mathcal{M}$ is $\mathcal{N}$-viable. Indeed, all receives from the same buffer are totally ordered by $\leq_{\mathbb{P}}$ when the

[1] This definition of $<_{\mathcal{N}}$ is tailored for many-to-one process networks, but for simplicity we have chosen not to mention the restriction in the definition. Note that $<_{\mathcal{N}}$ is a strict partial order.

process network is many-to-one, and the corresponding sends are ordered in the same way because of the buffer order. For example, the sequence shown in Figure 1a is mb-viable, but $p_1!p_2(m_1)\, p_2?p_1(m_1)\, p_2!p_1(m_2)\, p_0!p_1(m_0)\, p_1?p_0(m_0)$ is not.

For a process network $\mathcal{N}$ and a CFM $\mathcal{A}$ we write $\mathtt{msc}_{\mathcal{N}}(\mathcal{A}) = \{\mathtt{msc}(u) \mid u \in Tr_{\mathcal{N}}(\mathcal{A})\}$ for the set of MSCs associated with initial executions of $\mathcal{A}$. By Remark 2.11, the set $\mathtt{msc}_{\mathcal{N}}(\mathcal{A})$ consists only of $\mathcal{N}$-valid MSCs. The next definition introduces an equivalence relation $\equiv$ on CFM traces that is ubiquitous in this paper. Two traces are equivalent up to commuting adjacent actions that are neither performed by the same process, nor a matching send/receive pair:

▶ **Definition 2.12** (Equivalence $\equiv$)**.** *Two p2p-viable sequences $u, v \in Act^*$ are called* equivalent *if $\mathtt{msc}(u) = \mathtt{msc}(v)$ (up to isomorphism), and we write $u \equiv v$ in this case.*

▶ Remark 2.13. Two p2p-viable sequences are equivalent iff they have the same projection on each process.

▶ Remark 2.14. If $u, v \in Act^*$ are both $\mathcal{N}$-viable with $u \equiv v$, then $u \in Tr_{\mathcal{N}}(\mathcal{A})$ iff $v \in Tr_{\mathcal{N}}(\mathcal{A})$. However, $\equiv$ does not preserve $\mathcal{N}$-viability, e.g. $p!q(m)\, r!q(m)\, q?p(m) \equiv r!q(m)\, p!q(m)\, q?p(m)$, but the left-hand side is mb-viable while the right-hand side is not.

For the rest of the section $\mathcal{N} = (\mathsf{B}, \mathsf{bf})$ always refers to a process network. In order to be able to cope with partial executions we start by observing that unmatched sends to a buffer restrict the product of $\mathcal{N}$-viable sequences. Let u and v be two $\mathcal{N}$-viable sequences. The product $u *_{\mathcal{N}} v$ is defined if for every buffer $\mathsf{b} \in \mathsf{B}$, if there is an unmatched send to b in u, then there is no receive from b in v. When it is defined, $u *_{\mathcal{N}} v$ is equal to uv. Note that the partial binary operation $*_{\mathcal{N}}$ is associative. Moreover, if $u_0 *_{\mathcal{N}} \cdots u_i *_{\mathcal{N}} \cdots u_j *_{\mathcal{N}} u_{j+1} \cdots u_n$ is defined then $u_0 *_{\mathcal{N}} \cdots u_i *_{\mathcal{N}} u_{j+1} \cdots u_n$ is also defined, for every $i < j$. Note also that, when it is defined, the $*_{\mathcal{N}}$-product of two $\mathcal{N}$-viable sequences is $\mathcal{N}$-viable.

▶ **Definition 2.15** (Exchanges, synchronizability)**.**

1. *An $\mathcal{N}$-*exchange *is any $\mathcal{N}$-viable sequence $w \in S^* R^*$.*
2. *An $\mathcal{N}$-viable sequence u is called* $\mathcal{N}$-synchronous *if it is a $*_{\mathcal{N}}$-product of $\mathcal{N}$-exchanges. It is called* $\mathcal{N}$-synchronizable *if $u \equiv v$ for some $\mathcal{N}$-synchronous sequence v.*
3. *A CFM $\mathcal{A}$ is $\mathcal{N}$-synchronizable if all its traces $u \in Tr_{\mathcal{N}}(\mathcal{A})$ are $\mathcal{N}$-synchronizable.*

▶ Remark 2.16. The above definition of $\mathcal{N}$-synchronizability for $\mathcal{N} = \mathtt{mb}$ differs from the one initially used by [3, 7] and later called *weak-synchronizability* in [2] (mb-synchronizable here coincides with *strongly synchronizable* in [2]). An mb-viable sequence of actions u is *weakly-synchronizable* if it is is equivalent to a $*_{\mathtt{p2p}}$-product v of mb-exchanges. However, v is not required to be mb-viable. *Weak-synchronizability* yields more synchronizable traces, however some of them are spurious. In particular one cannot use the decompositions into exchanges from [3, 2] to check regular properties of executions, as we do in Section 4 later. Figure 1b shows an example distinguishing the definitions. The sequence there corresponds to a decomposition in exchanges according to [3, 2], but it is not mb-viable.

We end this section by a comparison between synchronizability for peer-to-peer semantics and mailbox semantics. These two notions are incomparable, in general. First, mb-synchronizability does not imply p2p-synchronizability simply because a system under mb-semantics has less executions than under p2p-semantics. Conversely, the following execution is mb-viable and p2p-synchronizable, but not mb-synchronizable (as we will see later the unmatched send makes it non-decomposable): $p!r(a)\, q!p(b)\, p?q(b)\, p!q(c)\, q?p(c)\, r!q(d)\, r?p(a)$. Finally, we note that p2p-synchronizability was shown to be undecidable in [2].

3 Reachability for mb-synchronizable systems

We start this section by showing that state reachability for mb-synchronizable CFMs is PSPACE-complete. The decidability (in exponential time) for mb-synchronizable CFMs can be already be inferred from [2] using the partial order logic LCPDL. The main point of this section is to introduce an automata-based approach to deal with mb-synchronizable CFMs. Although the set of mb-synchronous traces of a CFM is not regular in general, the projection of this set on (marked) send actions turns out to be regular. This crucial property is used later as a basic ingredient by our algorithm for deciding mb-synchronizability.

We start with an important observation saying that mb-synchronizability allows to focus on send actions. However, unmatched and matched sends need to be distinguished. So we introduce an extended alphabet $\overline{S} = \{\overline{s} \mid s \in S\}$. Sequences over $S \cup \overline{S}$ will be referred to as ms-sequences. For any mb-viable sequence u, we annotate every unmatched send $p!q(m)$ in u by $\overline{p!q(m)}$ and we denote by $\mathtt{marked}(u)$ the sequence obtained in this way. For example, for $u = p!q(m)p!r(m')r?p(m')$ we have $\mathtt{marked}(u) = \overline{p!q(m)}p!r(m')r?p(m')$. The *ms-sequence* $\mathtt{ms}(u)$ associated with an mb-viable sequence u is the projection of $\mathtt{marked}(u)$ on $S \cup \overline{S}$.

▶ **Lemma 3.1.**

1. *For any mb-exchanges u, v with $\mathtt{ms}(u) = \mathtt{ms}(u')$, we have $u \equiv u'$.*
2. *For any mb-exchange $u = vv'$ with $v \in S^*, v' \in R^*$, we define $\hat{u} = vv''$ with v'' obtained from v' by ordering the receives as their matching sends in v. Then $\hat{u}$ is mb-viable and $u \equiv \hat{u}$.*

Proof. For item 1, as $\mathtt{ms}(u) = \mathtt{ms}(u')$ and u, u' are both mb-viable, we get that for each process p, the sequence of receives by p in u and u', resp., are the same. We derive from $u, u' \in S^*R^*$ that u and u' have the same projection on each process, and thus $u \equiv u'$. For item 2 it is easy to check that $\hat{u}$ is mb-viable, hence $u \equiv \hat{u}$ by item 1. ◀

▶ Remark 3.2. It is worth noting that Lemma 3.1 does not hold anymore under p2p-semantics. For example, the two p2p-exchanges $u = p_1!p_2(a)\, p_3!p_2(b)\, p_2?p_3(b)\, p_2?p_1(a)$ and $\hat{u} = p_1!p_2(a)\, p_3!p_2(b)\, p_2?p_1(a)\, p_2?p_3(b)$ have the same marked sequence, but they are not equivalent. This is the main reason why our decidability results don't carry over to the p2p-semantics.

Executable mb-exchanges

We now show how to check if an ms-sequence corresponds to an executable mb-exchange of a CFM $\mathcal{A}$. Since we use the same construction also for the model-checking problem in Section 4 we give a more general formulation below.

Given an mb-viable sequence u and two sets $D, D' \subseteq \mathbb{P}$, we write $D \overset{u}{\rightsquigarrow} D'$ if no process from D receives any message in u, and D' contains D and those processes q such that u has some unmatched send to q. We refer to processes in D, D' as *deaf* processes. It is routinely checked that, for every mb-viable sequences $u_1, \ldots, u_n$, the product $u_1 *_{\mathtt{mb}} \ldots *_{\mathtt{mb}} u_n$ is defined iff $D_0 \overset{u_1}{\rightsquigarrow} D_1 \cdots \overset{u_n}{\rightsquigarrow} D_n$ for some sets $D_0, \ldots, D_n$.

▶ **Definition 3.3** (R-diamond)**.** *Let $\mathcal{A} = (L, S \cup \overline{S} \cup R, \rightarrow_{\mathcal{A}})$ be an LTS. We say that $\mathcal{A}$ is R-*diamond *if for all states $\ell, \ell' \in L$ and all* receives *$a, a' \in R$ performed by different processes, we have $\ell \xrightarrow{aa'}_{\mathcal{A}} \ell'$ iff $\ell \xrightarrow{a'a}_{\mathcal{A}} \ell'$.*

For any states ℓ, ℓ' of $\mathcal{A}$, sets $D, D' \subseteq \mathbb{P}$ and mb-viable sequence u, we write $(\ell, D) \overset{u}{\rightsquigarrow}_{\mathcal{A}} (\ell', D')$ if $\ell \xrightarrow{\mathtt{marked}(u)}_{\mathcal{A}} \ell'$ and $D \overset{u}{\rightsquigarrow} D'$. The next lemma shows how to adapt an R-diamond LTS to work on ms-sequences instead of mb-synchronous sequences (a similar idea appears in [12]):

▶ **Lemma 3.4.** *Assume that $\mathcal{A} = (L, S \cup \overline{S} \cup R, \to_{\mathcal{A}})$ is an R-diamond LTS. Then we can construct an LTS with ε-transitions $\mathcal{A}_{sync} = ((L \cup L^3) \times 2^{\mathbb{P}}, S \cup \overline{S}, \to_{sync})$ such that for any $v \in (S \cup \overline{S})^*$, states $\ell, \ell' \in L$, and sets $D, D' \subseteq \mathbb{P}$:*

$$(\ell, D) \xrightarrow{v}_{sync} (\ell', D') \quad \textit{iff} \quad \exists u \ \texttt{mb}\textit{-synchronous s.t. } v = \texttt{ms}(u) \textit{ and } (\ell, D) \overset{u}{\rightsquigarrow}_{\mathcal{A}} (\ell', D')$$

Proof. The LTS $\mathcal{A}_{sync}$ has the following transitions, for any $\ell, \ell' \in L$, $D, D' \subseteq \mathbb{P}$, $a \in S \cup \overline{S}$:

$$\begin{cases} (\ell, D) \xrightarrow{\varepsilon}_{sync} (\ell, \hat{\ell}, \hat{\ell}, D) & \text{for any } \hat{\ell} \in L \\ (\ell, \ell', \hat{\ell}, D) \xrightarrow{a}_{sync} (\ell_1, \ell'_1, \hat{\ell}, D) & \text{if } a = p!q(m), q \notin D, \ell \xrightarrow{a}_{\mathcal{A}} \ell_1, \ell' \xrightarrow{q?p(m)}_{\mathcal{A}} \ell'_1 \\ (\ell, \ell', \hat{\ell}, D) \xrightarrow{a}_{sync} (\ell_1, \ell', \hat{\ell}, D') & \text{if } a = \overline{p!q(m)}, \ell \xrightarrow{a}_{\mathcal{A}} \ell_1, D' = D \cup \{q\} \\ (\ell, \ell', \hat{\ell}, D) \xrightarrow{\varepsilon}_{sync} (\ell', D) & \text{if } \ell = \hat{\ell} \end{cases}$$

In other words, from a state $(\ell, D) \in L \times 2^{\mathbb{P}}$ the LTS $\mathcal{A}_{sync}$ first guesses a "middle" state $\hat{\ell} \in L$ for the current exchange, as the state reached after the sends. Then it switches to state $(\ell, \hat{\ell}, \hat{\ell}, D)$. The first component and the second component track sends and their matching receives (if matched) in a "synchronous" fashion. The LTS $\mathcal{A}_{sync}$ also guesses the end of the current mb-exchange, checking that the first component has reached the middle state $\hat{\ell}$ guessed originally. The claimed property of $\mathcal{A}_{sync}$ follows from Lemma 3.1 (2) and from $\mathcal{A}$ being R-diamond. ◀

Fix now a CFM $\mathcal{A}$. We abusively use the same notation $\rightsquigarrow_{\mathcal{A}}$ as above for LTS: for any global states $g, g' \in G$ of $\mathcal{A}$, sets $D, D' \subseteq \mathbb{P}$ and mb-viable sequence u, we write $(g, D) \overset{u}{\rightsquigarrow}_{\mathcal{A}} (g', D')$ if u labels an execution in $\mathcal{T}_{\texttt{mb}}(\mathcal{A})$ from the configuration $(g, \varepsilon^{\mathsf{B}})$ to some configuration $(g', (w_{\mathsf{b}})_{\mathsf{b} \in \mathsf{B}})$, and $D \overset{u}{\rightsquigarrow} D'$. We obtain from the previous lemma that:

▶ **Lemma 3.5.** *Let $\mathcal{A}$ be a CFM, $g, g' \in G$ two global states of $\mathcal{A}$, and $D, D' \subseteq \mathbb{P}$ two sets of processes. One can construct automata $\mathcal{B}, \mathcal{C}$ with $O(|G|^3 \times 2^{|\mathbb{P}|})$ states such that*

$$\begin{aligned} L(\mathcal{B}) &= \left\{ v \in (S \cup \overline{S})^* \mid \exists u \ \texttt{mb}\textit{-exchange s.t. } v = \texttt{ms}(u) \textit{ and } (g, D) \overset{u}{\rightsquigarrow}_{\mathcal{A}} (g', D') \right\}, \\ L(\mathcal{C}) &= \left\{ v \in (S \cup \overline{S})^* \mid \exists u \ \texttt{mb}\textit{-synchronous s.t. } v = \texttt{ms}(u) \textit{ and } (g, D) \overset{u}{\rightsquigarrow}_{\mathcal{A}} (g', D') \right\}. \end{aligned}$$

Proof. Assume that $\mathcal{A} = (\mathcal{A}_p)_{p \in \mathbb{P}}$. Let $\mathcal{Q}$ denote the asynchronous product $\prod_{p \in \mathbb{P}} \overline{\mathcal{A}}_p$, where each $\overline{\mathcal{A}}_p$ is the LTS obtained from $\mathcal{A}_p$ by adding a transition $\ell_p \xrightarrow{\overline{s}}_p \ell'_p$ for each transition $\ell_p \xrightarrow{s}_p \ell'_p$ with $s \in S$. Note that $\mathcal{Q}$ is R-diamond. Moreover, it is routinely checked that, for every mb-viable sequence u, the relation $\overset{u}{\rightsquigarrow}_{\mathcal{A}}$ coincides with the relation $\overset{u}{\rightsquigarrow}_{\mathcal{Q}}$.

For $\mathcal{C}$ we take the automaton $\mathcal{Q}_{sync}$ constructed according to Lemma 3.4, and set the initial state to (g, D) and the final state to (g', D'). For $\mathcal{B}$, we need to tinker a bit with $\mathcal{Q}_{sync}$ to ensure that we read only one exchange. So we remove all transitions from/to states in $L \times 2^{\mathbb{P}}$ except the transitions from (g, D), which we set as initial, and the transitions to (g', D'), which we set as final. If $(g, D) = (g', D')$ then we make two different states for the initial and the final one. ◀

Using Lemma 3.5 we establish the upper bound of the global-state reachability problem for mb-synchronizable CFMs (the lower bound is straightforward). By global-state reachability we mean the existence of a reachable configuration with a specified global state. Decidability was shown in [7] for weak-synchronizability (correcting the proof in [3]) *and* assuming a uniform bound on the size of exchanges.

▶ **Theorem 3.6.** *The global-state reachability problem for* mb*-synchronizable CFMs is* PSPACE*-complete.*

Proof. Note first that if $\mathcal{A}$ is a CFM and u, v two mb-viable sequences u, v with $u \equiv v$ then $c_{in} \xrightarrow{u}_{\mathcal{A}} c$ implies that $c_{in} \xrightarrow{v}_{\mathcal{A}} c'$ for some c' with the same global state as c. Since we assume that the CFM is mb-synchronizable we can choose v to be mb-synchronous. Thus we can use automaton $\mathcal{C}$ from Lemma 3.5 to show the upper bound. This automaton can clearly be constructed on-the-fly in polynomial space.

For the lower bound we reduce from the problem of intersection of NFA. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be NFA over the alphabet Σ. We use processes $p_1, \ldots, p_n$ where each p_i simulates $\mathcal{A}_i$. Process p_1 starts by guessing a letter a of Σ, making a transition on a and sending a to p_2. Afterwards each process p_i receives a letter a from p_{i-1}, makes a transition on a, then sends a to p_{i+1}. Back again at p_1, the procedure restarts. Figure 2 shows the principle.

Upon reaching a final state, p_1 can send message `accept` to p_2 and then stop. If p_i receives `accept` from p_{i-1} while being in a final state, it relays `accept` to p_{i+1}, and then stops.

One can see that every trace of the CFM is mb-synchronizable, as every message is in its own exchange. Moreover, the global-state $(\texttt{accept})_{p \in \mathbb{P}}$ is reachable if and only if the intersection of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is non-empty. ◀

4 Model-checking regular properties

In this section we introduce a class of properties against which we can verify mb-synchronizable CFMs. We look for regular properties P over the alphabet $S \cup R \cup \overline{S}$, so we exploit the marked sends to refer (indirectly) to messages. The model-checking problem we consider is the following:

CFM-VS-REGULAR PROPERTY
INPUT: mb-synchronizable CFM $\mathcal{A}$, regular property $P \subseteq (S \cup \overline{S} \cup R)^*$.
OUTPUT: Yes if for every mb-synchronous trace $u \in Tr_{\texttt{mb}}(\mathcal{A})$ we have $\texttt{marked}(u) \in P$.

The properties we consider are regular, R-closed subsets of $(S \cup \overline{S} \cup R)^*$:

▶ **Definition 4.1** (R-closed properties). *Let $\equiv_R$ be the reflexive-transitive closure of the relation consisting of all pairs $(u\,a\,b\,v, u\,b\,a\,v)$ with $u, v \in (S \cup \overline{S} \cup R)^*$, $a, b \in R$, and a, b performed by distinct processes. A property $P \subseteq (S \cup \overline{S} \cup R)^*$ is called* R-closed *if it is closed under $\equiv_R$ (i.e., for any $u \equiv_R v$ we have $u \in P$ iff $v \in P$).*

As an example, we can consider a system with a central process c and a set of orbiting processes $p_1, \ldots, p_n$. The central process gives tasks to the orbiting processes, and they send back their results. We can state a property expressing a round-based behavior for c: it sends tasks to orbiting processes, and if a process p_i does not send back to c in the next round, it will not participate in further rounds anymore. The opposite property consists of all sequences from $A^* {S_c}^* c!p_i(m) {S_c}^* R^+ (\bigcup_{j \neq i} S_{p_j})^+ R^+ {S_c}^+ R^* A^* p_i!c(m') A^*$ for some i and m, m', and $A = S \cup \overline{S} \cup R$. As the above property is R-closed, its complement is too.

We will show that if the regular property is R-closed then the model-checking problem stated above is PSPACE-complete. Before that recall that both being mb-viable and being mb-synchronous (assuming mb-viable) are non regular properties. However, it is not necessary to be able to express the above, as we will apply the property to mb-synchronous traces of CFM. The next lemma is similar to Lemma 3.4:

▶ **Lemma 4.2.** *Let $P \subseteq (S \cup \overline{S} \cup R)^*$ be regular and R-closed. Then the set*

$$Sync(P) = \{v \in (S \cup \overline{S})^* \mid \exists u \text{ } \texttt{mb}\text{-synchronous} \text{ s.t. } v = \texttt{ms}(u) \text{ and } \texttt{marked}(u) \in P\}$$

is regular. If P is given by an R-diamond NFA with n states, then we can construct an NFA for $Sync(P)$ with $O(n^3 \cdot 2^{|\mathbb{P}|})$ states.

Proof. Let P be given by an R-diamond NFA $\mathcal{P} = (L, S \cup \overline{S} \cup R, \rightarrow_{\mathcal{P}}, \ell_0, F)$ with n states. We may assume w.l.o.g. that $\mathcal{P}$ contains no ε-transition. Consider the LTS with ε-transitions $\mathcal{P}_{sync}$ obtained from Lemma 3.4. Recall that this LTS has $O(n^3 \times 2^{|\mathbb{P}|})$ states. As NFA for Sync(P), we take $\mathcal{P}_{sync}$, with $(\ell_0, \emptyset)$ as initial state, and $F \times 2^{\mathbb{P}}$ as final states. ◀

▶ **Theorem 4.3.** *The* CFM-VS-REGULAR PROPERTY *problem is* PSPACE*-complete if the property is R-closed. There exist properties that are not R-closed for which the problem is undecidable.*

Proof. For the upper bound, consider an mb-synchronizable CFM $\mathcal{A} = (\mathcal{A}_p)_{p \in \mathbb{P}}$ and an R-closed regular property $P \subseteq (S \cup \overline{S} \cup R)^*$ given by an NFA $\mathcal{P}$. Since P is R-closed, its complement P^{co} is also R-closed. As in the proof of Lemma 3.5, let $\mathcal{Q}$ denote the asynchronous product $\prod_{p \in \mathbb{P}} \overline{\mathcal{A}}_p$, where each $\overline{\mathcal{A}}_p$ is the LTS obtained from $\mathcal{A}_p$ by adding a transition $\ell_p \xrightarrow{\overline{s}}_p \ell'_p$ for each transition $\ell_p \xrightarrow{s}_p \ell'_p$ with $s \in S$. Note that $\mathcal{Q}$ is R-diamond, so its language $Q = L(\mathcal{Q})$ is R-closed. We derive that $Q \cap P^{co}$ is R-closed. It is routinely checked that $(\mathcal{A}, \mathcal{P})$ is a positive instance of CFM-VS-REGULAR PROPERTY iff the set Sync$(Q \cap P^{co})$, as defined in Lemma 4.2, is empty. To derive the PSPACE upper bound from this lemma, we still need to provide an R-diamond NFA for $Q \cap P^{co}$. This R-diamond NFA is simply the synchronous product of $\mathcal{Q}$ and the minimal automaton of P^{co}. The latter is R-diamond since P^{co} is R-closed, and it can be constructed on-the-fly in polynomial space from $\mathcal{P}$. Now it suffices to check emptiness of the NFA for Sync$(Q \cap P^{co})$ from Lemma 4.2. The lower bound is again straightforward.

For the undecidability of model-checking a property that is not R-closed we use a straightforward reduction from PCP. Let $(u_i, v_i)_{i=1\dots k}$ be an instance of PCP over the binary alphabet $\{\texttt{0}, \texttt{1}\}$. We can have three processes p, U, V and process p who sends, in rounds, some pair (u_i, v_i) to U and V, resp. That is, p sends u_i (v_i, resp.) letter by letter to U (V, resp.). The processes U and V do nothing except receiving whatever p sends to them.

There is a solution to the given PCP instance iff there is a trace consisting of a single fully matched mb-exchange where U and V perform the same receives in lock-step. So we take as property P the regular language $P = (S \cup \overline{S} \cup R)^* \setminus P^{co}$ where $P^{co} = S^*\{U?p(\texttt{0})V?p(\texttt{0}), U?p(\texttt{1})V?p(\texttt{1})\}^*$. ◀

Comparing p2p and mb semantics

Given a protocol that was designed for p2p communication, it can be useful to know whether the protocol can be also deployed under mailbox communication. We call this property mailbox-similarity:

▶ **Definition 4.4** (Mailbox-similarity). *A p2p-viable sequence of actions u is called* mailbox-similar *if there exists some* mb*-viable sequence v such that $u \equiv v$. A CFM $\mathcal{A}$ is called* mailbox-similar *if every trace from $Tr_{\texttt{p2p}}(\mathcal{A})$ is mailbox-similar.*

Equivalently, a CFM $\mathcal{A}$ is mailbox-similar if every MSC from $\texttt{msc}_{\texttt{p2p}}(\mathcal{A})$ is mb-valid. Unsurprisingly, as it is often the case under p2p semantics, mailbox-similarity is undecidable without further restrictions:

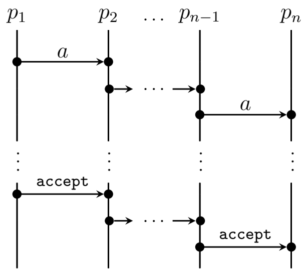


Figure 2 The MSC of a trace of the CFM for automata intersection.

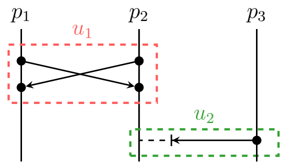


Figure 3 MSC of $u = p_2!p_1(m_1)\, p_1!p_2(m_2)\, p_1?p_2(m_1)\, p_2?p_1(m_2)\;\; p_3!p_2(m_3)$, with the two SCCs of its communication graph. Note that $1 \preceq^u_{\mathtt{mb}} 2$, but neither $1 \preceq^u_{\mathtt{p2p}} 2$ nor $2 \preceq^u_{\mathtt{p2p}} 1$ holds.

▶ **Lemma 4.5.** *The question whether a given CFM is mailbox-similar is undecidable.*

In the remainder of this section, we show that mailbox-similarity becomes decidable if we assume that the CFM is $\mathtt{mb}$-synchronizable. Recall that the latter means that every trace from $Tr_{\mathtt{mb}}(\mathcal{A})$ is $\mathtt{mb}$-synchronizable.

The next lemma shows how to check that two positions in an $\mathtt{mb}$-synchronous sequence u are causally-ordered, i.e., there is some $(<_{\mathrm{hb}} \cup <_{\mathtt{mb}})$-path between these positions (as usual, this refers to a path between associated events in $\mathtt{msc}(u)$). We mark these positions by using a “tagged” alphabet $\Sigma = (S \cup \overline{S} \cup R) \times \{\circ, \bullet\}$.

▶ **Lemma 4.6.** *We can construct an R-diamond automaton $\mathcal{D}$ with $O(|\mathbb{P}|)$ states over the alphabet Σ such that for every $\mathtt{mb}$-synchronous sequence $u \in Act^*$ and every positions $i < j$ of u such that $u[i]$ and $u[j]$ are in S, there is a $(<_{hb} \cup <_{\mathtt{mb}})$-path from $u[i]$ to $u[j]$ iff $\mathcal{D}$ accepts the word $\mathtt{marked}(u)$ tagged by $\bullet$ at i and j and by $\circ$ elsewhere.*

Proof. Recall that $\leq_{\mathrm{hb}} = (<_{\mathbb{P}} \cup \mathrm{msg})^*$ is the happens-before order. The automaton $\mathcal{D}$ will guess a $(<_{\mathbb{P}} \cup \mathrm{msg} \cup <_{\mathtt{mb}})$-path from $u[i]$ to $u[j]$. It will actually use only send actions of $\mathtt{marked}(u)$, relying on the fact that u is $\mathtt{mb}$-synchronous. That is, $\mathcal{D}$ guesses a subsequence of positions $i_1 < \cdots < i_t$ of u, with each $u[i_k] \in S$, as described in the following. Let $i_0 = i$ and $i_{t+1} = j$. We have three cases, and $\mathcal{D}$ guesses in which case we are:

- $u[i_k], u[i_{k+1}]$ are performed by the same process p. After i_k the automaton $\mathcal{D}$ remembers the pair $(<_{\mathbb{P}}, p)$ until it guesses i_{k+1}.
- $u[i_k], u[i_{k+1}]$ are both sends to the same process p, and $u[i_k]$ is matched. After i_k the automaton $\mathcal{D}$ remembers $(<_{\mathtt{mb}}, p)$ until it guesses i_{k+1}.
- $u[i_k]$ is matched, its receive $u[h]$ is performed by the same process p as $u[i_{k+1}]$, and $h < i_{k+1}$. After i_k the automaton $\mathcal{D}$ remembers the pair (msg, S, p). After the next receive action, $\mathcal{D}$ changes its state to (msg, R, p) until it guesses i_{k+1}. The assumption that u is $\mathtt{mb}$-synchronous guarantees that the receive $u[h]$ matched with $u[i_k]$ has already occurred when $\mathcal{D}$ guesses i_{k+1}.

By construction, if $\mathcal{D}$ accepts $\mathtt{marked}(u)$, then we have a $(<_{\mathrm{hb}} \cup <_{\mathtt{mb}})$-path from $u[i]$ to $u[j]$, with $i < j$ the two positions tagged by $\bullet$ in $\mathtt{marked}(u)$.

For the left-to-right implication, assume that $u[i]$ and $u[j]$ are in S and that we have a $(<_{\mathrm{hb}} \cup <_{\mathtt{mb}})$-path from $u[i]$ to $u[j]$. This path is a sequence $i = i_0 < i_1 \cdots < i_t < i_{t+1} = j$ of positions of u, such that each pair of consecutive indices is related by $<_{\mathbb{P}}$, $<_{\mathtt{mb}}$ or msg. Moreover, we may assume w.l.o.g. that there are no two consecutive $<_{\mathbb{P}}$-arcs on this path. If the path contains only $<_{\mathbb{P}}$ and $<_{\mathtt{mb}}$-arcs, then $\mathcal{D}$ applies one of the first two rules above. Consider now a msg-arc $(u[i_k], u[i_{k+1}])$. As $u[i_{k+1}]$ is a receive, we get that $u[i_{k+1}] <_{\mathbb{P}} u[i_{k+2}]$.

Moreover, $u[i_{k+2}]$ is a send since there are no two consecutive $<_{\mathbb{P}}$-arcs on the path. So $\mathcal{D}$ can apply the third rule to go from i_k to i_{k+2}. We get that $\mathcal{D}$ accepts the word $\mathtt{marked}(u)$ tagged by $\bullet$ at i and j and by $\circ$ elsewhere. The number of states of $\mathcal{D}$ is $4 * |\mathbb{P}| + 2$ (2 for initial/final state). ◀

▶ **Lemma 4.7.** *For any receive action $r \in R$, we can construct an R-diamond automaton $\mathcal{P}_r$ with $O(|\mathbb{P}|)$ states over the alphabet $(S \cup \overline{S} \cup R)$ such that for every* mb*-synchronous sequence u, it holds that $u\,r$ is* p2p*-viable and not mailbox-similar iff $\mathcal{P}_r$ accepts $\mathtt{marked}(u)$.*

Proof sketch. Consider a receive action $r = q?p(m)$. Let W_r denote the set of words $w \in \Sigma^*$ such that w contains exactly two positions $i < j$ tagged by $\bullet$, $w[i]$ is an unmatched send to q, $w[j]$ is $\overline{p!q(m)}$, and no $w[h]$ with $h < j$ is an unmatched send from p to q. It is easily seen that W_r is recognized by an R-diamond NFA $\mathcal{W}_r$ with three states. Let $\mathcal{E}_r$ denote the synchronous product of $\mathcal{W}_r$ and the R-diamond automaton $\mathcal{D}$ from Lemma 4.6. The desired automaton $\mathcal{P}_r$ is obtained from $\mathcal{E}_r$ by untagging it, that is, by replacing each tagged action $(a, t) \in \Sigma$ by a. As $\mathcal{E}_r$ is R-diamond, so is $\mathcal{P}_r$. By construction, $\mathcal{P}_r$ satisfies the lemma condition. The details can be found in the full version of the paper [6]. ◀

We derive from the previous lemma that mailbox-similarity can be solved in PSPACE for mb-synchronizable CFMs. The proof uses Lemma 4.2 and is similar to the proof of Theorem 4.3.

▶ **Theorem 4.8.** *The question whether a given* mb*-synchronizable CFM is mailbox-similar is* PSPACE*-complete.*

5 Checking mb-synchronizability

In this section we show our main result, namely an algorithm to know if a CFM is mb-synchronizable. As a side result we obtain optimal complexity bounds for some problems considered in [7, 12].

The high-level schema of the algorithm is to look for a minimal witness for non-mb-synchronizability. This amounts to searching for an mb-synchronous trace that violates mb-synchronizability after adding one (receive) action. Of course, we need Theorem 3.6 to guarantee that the mb-synchronous trace is executable. In addition, we have to detect the violation of mb-synchronizability, and for this we need to determine if an exchange is non-decomposable into smaller exchanges. Section 5.1 shows automata for non-decomposable exchanges, and in Section 5.2 we present the algorithm that finds minimal witnesses.

5.1 Automata for atomic exchanges

In this section we consider sequences of actions that cannot be split into smaller pieces without separating messages [11, 12]. We introduce these notions for arbitrary many-to-one process networks $\mathcal{N}$. Later we will fix $\mathcal{N} = \mathtt{mb}$ since reachability over synchronizable sequences is decidable in this setting.

▶ **Definition 5.1** (Atomic sequences). *An $\mathcal{N}$-viable sequence $u \in Act^*$ is* $\mathcal{N}$-atomic *(or* atomic *for short) if $u \equiv v *_{\mathcal{N}} w$ with v, w both $\mathcal{N}$-viable implies that one of v, w is empty.*

To check atomicity we can use a graph criterium introduced already in [13] (see also [11]), that is similar to the notion of conflict graph used in [3]:

▶ **Definition 5.2** (Communication graph). *Let u be an $\mathcal{N}$-viable sequence, and $\mathcal{M} = \texttt{msc}(u)$. The* $\mathcal{N}$-communication graph *of u is the directed graph $H_\mathcal{N}(u) = (V, E)$ where V is the set of all events of $\mathcal{M}$ and the edges are defined by $(e, e') \in E$ if $e <_\mathbb{P} e'$ or $e <_\mathcal{N} e'$ or $\{(e, e'), (e', e)\} \cap \text{msg} \neq \emptyset$.*

The right part of Figure 4 shows (partly) the communication graph of the MSC in the left part. The cycle witnesses that the MSC is $\mathcal{N}$-atomic for $\mathcal{N} \in \{\texttt{mb}, \texttt{p2p}\}$, according to the next lemma.

▶ **Lemma 5.3.** *Let $u \in Act^*$ be a $\mathcal{N}$-viable sequence and $H_\mathcal{N}(u)$ the $\mathcal{N}$-communication graph of $\texttt{msc}(u)$. Then u is $\mathcal{N}$-atomic if and only if $H_\mathcal{N}(u)$ is strongly connected.*

From Lemma 5.3 we can infer a decomposition of any trace in atomic subsequences that is unique up to permuting adjacent atomic sequences that are not ordered in the sense of the next definition:

▶ **Definition 5.4** (Skeleton). *Let u be a $\mathcal{N}$-viable sequence with $\mathcal{M} = \texttt{msc}(u)$ and $H_\mathcal{N}(u)$ be the $\mathcal{N}$-communication graph of $\mathcal{M}$. Fix some arbitrary topological indexing $\{1, \ldots, n\}$ of the SCCs of $H_\mathcal{N}(u)$. We define the* skeleton *of u as $\texttt{skel}(u) = (\{1, \ldots, n\}, \preceq^u_\mathcal{N})$, where $\preceq^u_\mathcal{N}$ is the partial order induced by setting $i \prec^u_\mathcal{N} j$ for $1 \leq i < j \leq n$ if there is some $<_\mathbb{P}$-arc or some $\texttt{mb}$-arc in $H_\mathcal{N}(u)$ from the SCC with index i to the SCC with index j.*

▶ Remark 5.5. Assume that $u = u_1 *_\mathcal{N} \cdots *_\mathcal{N} u_n$ where each u_i is $\mathcal{N}$-atomic and non-empty, and we index the SCCs according to the order of the u_i. Then we obtain $\texttt{skel}(u) = (\{1, \ldots, n\}, \preceq^u_\mathcal{N})$ with $i \prec^u_\mathcal{N} j$ if either both u_i and u_j contain some actions on the same process; or they both contain some send to the same buffer, with the one in u_i being matched. See Figure 3 for an example.

▶ **Lemma 5.6.** *Let u be an $\mathcal{N}$-viable sequence. Then there exist some $\mathcal{N}$-atomic non-empty sequences $u_1, \ldots, u_k$ such that $u \equiv u_1 *_\mathcal{N} \ldots *_\mathcal{N} u_k$. Such a decomposition into $\mathcal{N}$-atomic non-empty sequences is unique up to the partial order $\preceq^u_\mathcal{N}$ of $\texttt{skel}(u)$.*

Throughout the remaining of this section we fix $\mathcal{N} = \texttt{mb}$. We will show now a simple, automaton-compatible condition to certify that an ms-sequence $v = \texttt{ms}(u)$ corresponds to an $\texttt{mb}$-atomic exchange u. First we note that, in order for the communication graph $H_\texttt{mb}(u)$ to be strongly connected, there must exist for every process p that is active in u some path from the last action of p to the first action of p (if there are at least two actions of p in u). A process p is called *active* in u if there is at least some action performed by p in u (resp., if $v = \texttt{ms}(u)$ contains either a send performed by p, or a matched send to p). We look for such a path for every active process and then we need to connect all such paths together.

Let $u \in Act^*$ be an $\texttt{mb}$-exchange. For some suitable integer n we define a labeling of $v = \texttt{ms}(u)$ as an injective mapping $\pi : \{0, \ldots, n\} \to \{1, \ldots, |v|\}$ where $\pi(i) = j$ means that position j of v is labeled by i. We say that π is a *well-labeling* of v (of size n) if, for every $0 \leq i < n$:

- either $\pi(i) < \pi(i+1)$ and, for some process p:
  $v[\pi(i)]$ and $v[\pi(i+1)]$ are both sends by p, or
  $v[\pi(i)]$ and $v[\pi(i+1)]$ are both sends to p, with $v[\pi(i)]$ matched (*direct arc*)
- or $v[\pi(i)]$ is a send by p and $v[\pi(i+1)]$ is a matched send to p (*indirect arc*).

An example of such labeling is shown in Figure 4. Informally, one can see the two types of arcs between positions of v as:

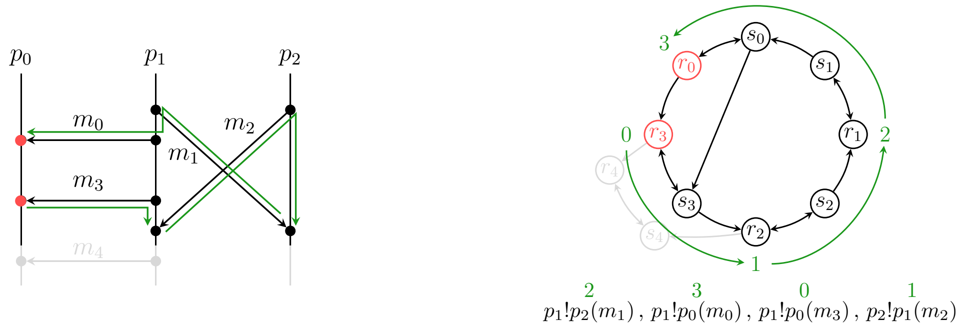


Figure 4 A well-labeling of the ms-sequence bottom right, witnessing a path in the communication graph of the MSC left, from the last to the first event of process p_0. The s_i and r_i vertices of the communication graph correspond respectively to the send and receive of message m_i.

- A direct arc between two sends corresponds to the process order $\leq_{\mathbb{P}}$ or the mailbox order $\leq_{\mathtt{mb}}$ in $\mathtt{msc}(u)$. For example, we have a direct arc from position 2 to 3 in Figure 4.
- An indirect arc between two sends stems from composing edges of the communication graph $H_{\mathtt{mb}}(u)$ that involve a receive event. An indirect arc is specific to $\mathtt{mb}$-exchanges: in $H_{\mathtt{mb}}(u)$ we can go from the event of $v[i]$ to the receive associated with the event of $v[j]$ (since u is an $\mathtt{mb}$-exchange this receive is after $v[i]$), and then follow the message edge backwards to the event of $v[j]$. For example, we have an indirect arc from position 1 to 2 in Figure 4.

▶ **Lemma 5.7.** *Let u be an $\mathtt{mb}$-exchange with $\mathcal{M} = \mathtt{msc}(u)$, and $v = \mathtt{ms}(u)$. There is a path in the communication graph $H_{\mathtt{mb}}(u)$ from the event of $\mathcal{M}$ corresponding to $v[i]$ to the event corresponding to $v[j]$ if and only if there is a well-labeling of v starting at i and ending at j.*

Proof. For the right-to-left direction, let π be a well-labeling of v starting at i and ending at j. As π is a well-labeling, there is a path in $H_{\mathtt{mb}}(u)$ from the event corresponding to $v[\pi(k)]$ to the one of $v[\pi(k+1)]$, for every k in the domain of π. Each such path is either a direct edge, or consists of two edges, as explained before the statement of the lemma in the main body.

For the left-to-right direction, we suppose there is a path Π in $H_{\mathtt{mb}}(u)$ from the event of $v[i]$ to the event of $v[j]$. We construct a labeling π of v that starts at i and ends at j, by labelling the positions of v that correspond to the events of Π with their respective rank on Π. Suppose that n positions are labeled and let $0 \leq k < n$. We show the existence of an arc from $\pi(k)$ to $\pi(k+1)$, which is either direct or indirect. There are three cases:

- There is no receive between the event of $v[\pi(k)]$ and the one of $v[\pi(k+1)]$ on Π. Thus $v[\pi(k)]$, $v[\pi(k+1)]$ are consecutive on Π and are either ordered by $<_{\mathbb{P}}$ or by $<_{\mathtt{mb}}$. This gives a direct arc from $\pi(k)$ to $\pi(k+1)$.
- Between the event of $v[\pi(k)]$ and the one of $v[\pi(k+1)]$ we see on Π the receive matching $v[\pi(k)]$ before the receive matching $v[\pi(k+1)]$. Note that both receives must be on the same process (as all receives between $v[\pi(k)]$ and $v[\pi(k+1)]$), so they are ordered by $<_{\mathbb{P}}$. Thus, the events of $v[\pi(k)]$ and $v[\pi(k+1)]$, respectively, are ordered by $<_{\mathtt{mb}}$. This gives a direct arc from $\pi(k)$ to $\pi(k+1)$.
- Between the event of $v[\pi(k)]$ and the one of $v[\pi(k+1)]$ we have on Π the receive matching the event of $v[\pi(k+1)]$ on the same process as the event of $v[\pi(k)]$. This gives an indirect arc from $\pi(k)$ to $\pi(k+1)$. ◀

▶ Remark 5.8. In Lemma 5.7, we only talk about send actions. If we are interested in a path in $H_{\mathtt{mb}}(u)$ to a receive action, we just need to exhibit the path to its corresponding send action.

We can infer a bound on the size of well-labelings, using the pigeonhole principle on the direct arcs and indirect arcs going through each process.

▶ **Lemma 5.9.** *Let $u \in Act^*$ be an* mb*-exchange and $v = \mathtt{ms}(u)$. If there is a path in the communication graph $H_{\mathtt{mb}}(u)$ between $v[i]$ and $v[j]$ then there is a well-labeling of $\mathtt{ms}(u)$ starting at position i and ending at position j of size at most $|\mathbb{P}|^2 + |\mathbb{P}|$.*

We construct now two kinds of automata, both working on ms-sequences $v = \mathtt{ms}(u)$. Automaton $\mathcal{B}_p$ will check for a process p that is active in u, that all actions performed by p are on a cycle in $H_{\mathtt{mb}}(u)$. Automaton $\mathcal{B}_{all}$ will check that all actions of active processes in u appear together on a cycle in $H_{\mathtt{mb}}(u)$, by looking for a cycle going through all active processes at least once. Finally we take the product of all automata $\mathcal{B}_p$ such that p is active and the automaton $\mathcal{B}_{all}$. The resulting automaton has $|\mathbb{P}|^{O(|\mathbb{P}|^3)}$ states and verifies the following property: for every mb-exchange u, it holds that u is atomic iff $\mathtt{ms}(u)$ is accepted by the automaton. By taking the product of this last automaton with the automaton verifying that the ms-sequence corresponds to an mb-exchange (see Lemma 3.5), we immediately get:

▶ **Lemma 5.10.** *Let $\mathcal{A}$ be a CFM, g, g' two global states of $\mathcal{A}$ and $D, D' \subseteq \mathbb{P}$. One can construct an automaton $\mathcal{B}$ with $O(|G|^3 \cdot |\mathbb{P}|^{O(|\mathbb{P}|^3)})$ states, such that*

$$L(\mathcal{B}) = \left\{ v \in (S \cup \overline{S})^* \mid \exists u \text{ atomic } \mathtt{mb}\text{-exchange } \text{ s.t. } v = \mathtt{ms}(u) \text{ and } (g, D) \overset{u}{\rightsquigarrow}_{\mathcal{A}} (g', D') \right\}$$

5.2 Verifying mb-synchronizability

To check mb-synchronizability we look for an mb-viable trace that is not equivalent to a $*_{\mathtt{mb}}$-product of mb-exchanges. Such a *witness* u must contain some atomic factor v that is not equivalent to an mb-exchange. In other words, $u \equiv u' *_{\mathtt{mb}} v *_{\mathtt{mb}} u''$ for some u', u'', with $v' \notin S^* R^*$ for every $v \equiv v'$. It is enough to reason on atomic factors, since for any exchange u where $u \equiv u_1 *_{\mathtt{mb}} \ldots *_{\mathtt{mb}} u_n$ with each u_i atomic, all factors u_i are also exchanges. Note that an atomic v is not equivalent to an mb-exchange iff some process in v does a send after a receive.

The next lemmas refer to the structure of *minimal witnesses* for non-mb-synchronizability.

▶ **Lemma 5.11.** *Let $u = v\,r$ be an* mb*-viable sequence with $r \in R$. There exist* mb*-atomic non-empty sequences $v_1, \ldots, v_n$ and indices $1 \le i < j \le n$ such that (1) $v \equiv v_1 *_{\mathtt{mb}} \cdots *_{\mathtt{mb}} v_n$, and (2) $u \equiv v_1 *_{\mathtt{mb}} \cdots *_{\mathtt{mb}} v_{i-1} *_{\mathtt{mb}} w *_{\mathtt{mb}} v_{j+1} *_{\mathtt{mb}} \cdots *_{\mathtt{mb}} v_n$ with $w = (v_i *_{\mathtt{mb}} \cdots *_{\mathtt{mb}} v_j)\,r$ being* mb*-atomic.*

▶ **Lemma 5.12.** *Let $u = v\,r$ be an* mb*-viable sequence with $r \in R$, such that v is not* mb*-atomic. We denote by s the send event matched with r in u, and by q the process of r. Then u is* mb*-atomic iff for every decomposition $v \equiv v_1 *_{\mathtt{mb}} \cdots *_{\mathtt{mb}} v_n$ with v_i* mb*-atomic for all i:
(1) v_1 contains s or some unmatched send to process q, and (2) v_n contains s or some action performed by process q.*

An example of such a decomposition is shown in Figure 5.

▶ **Lemma 5.13.** *Let $u = v\,r$ be* mb*-viable with $r \in R$ and v is* mb*-synchronizable. Let also s be the send matching r in u, and q the process doing r. Then u is not* mb*-synchronizable iff there exist $(v_i)_{i=1}^n$ with $v \equiv v_1 * \cdots * v_n$, indices $1 \le i_1 < \cdots < i_k \le n$, and $p \in \mathbb{P}$ s.t.:*

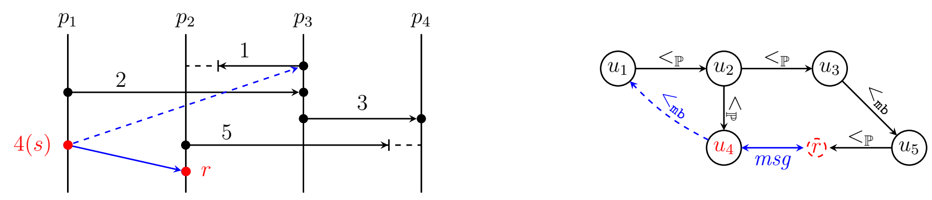


Figure 5 The MSC of an atomic sequence. It is not $\mathtt{mb}$-synchronizable by Lemma 5.13, each u_i consists of message i, the indices are $(1, 2, 3, 5)$, and $m = 2$.

1. *Each v_i is $\mathtt{mb}$-atomic.*
2. *For every $1 \leq j < k$ we have $i_j \prec^v_{\mathtt{mb}} i_{j+1}$.*
3. *v_{i_1} contains s or some unmatched send to process q; v_{i_k} contains s or some action performed by process q.*
4. *There exists $1 \leq m < k$ such that v_{i_m} contains a receive by p and $v_{i_{m+1}}$ a send by p.*

Note that while we can guess $\mathtt{mb}$-synchronous sequences without storing messages (Lemma 3.5), we need to be careful when guessing u in Lemma 5.13 so that it is $\mathtt{mb}$-viable. E.g., by reversing message 2 in Figure 5 the sequence becomes non-$\mathtt{mb}$-viable.

▶ **Lemma 5.14.** *Let $u = v\,r$ be a $\mathtt{p2p}$-viable sequence with $r \in R$ and v $\mathtt{mb}$-viable. Let q be the process performing r. Then u is equivalent to an $\mathtt{mb}$-viable sequence if and only if there is no non-empty $(<_{hb} \cup <_{\mathtt{mb}})$-path from $v[i]$ to $v[j]$ for some $i < j$ such that $v[i]$ is an unmatched send to q and $v[j]$ is the send matching r in u.*

The next lemma shows how to check the existence of a $(<_{\mathrm{hb}} \cup <_{\mathtt{mb}})$-path between two positions of an ms-sequence, using the automaton from Lemma 4.6.

▶ **Lemma 5.15.** *One can construct an automaton $\mathcal{D}$ with $O(|\mathbb{P}|)$ states over the alphabet $(S \cup \overline{S}) \times \{\circ, \bullet\} \cup \{\#\}$ with the following properties:*
1. *$\mathcal{D}$ accepts only words from $(\Sigma^*\#)^*$ containing exactly two positions in $(S \cup \overline{S}) \times \{\bullet\}$.*
2. *For every $u = u_1 *_{\mathtt{mb}} \ldots *_{\mathtt{mb}} u_n$ $\mathtt{mb}$-viable, with each u_i an exchange, $\mathcal{D}$ accepts tagged $v = \mathit{ms}(u_1)\#\ldots\#\mathit{ms}(u_n)$ iff, there is a $(<_{hb} \cup <_{\mathtt{mb}})$-path from $u[i]$ to $u[j]$, where $i < j$ are the positions of u corresponding to the positions tagged by $\bullet$ in v.*

We have now all ingredients to show our main result. We use Lemma 5.13 to guess the witness sequence, exchange by exchange, and to be sure that the sequence is $\mathtt{mb}$-viable we rely on Lemmas 5.14 and 5.15, complementing the automaton on-the-fly. The lower bound is obtained, as before, by reduction from the intersection emptiness problem for finite automata.

▶ **Theorem 5.16.** *The question whether a CFM is $\mathtt{mb}$-synchronizable is* PSPACE*-complete.*

Proof. For the upper bound we use Lemma 5.13 to guess a minimal non-$\mathtt{mb}$-synchronizable sequence $u = v\,r$. Recall that q is the process executing r, and s the matching send of r in u. First we rely on the automaton of Lemma 5.10 in order to guess the atomic exchanges v_i composing v on-the-fly. At the same time we guess the subsequence of indices $i_1 < \cdots < i_k$ and the events that witness that $i_j \prec^v_{\mathtt{mb}} i_{j+1}$ (cf. Definition 5.4).

We keep record of the current pair (g, D), where g is a global state of the CFM and D a set of deaf processes, as we guess each v_i, to check that the sequence v labels an execution. When we process v_{i_k}, we remember its alphabet over $S \cup \overline{S}$ until we guess $v_{i_{k+1}}$, and check that $i_k \prec^v_{\mathtt{mb}} i_{k+1}$ (cf. Remark 5.5). We also guess m as of item (5) in Lemma 5.13, and check (5). After we have done v_n, we must have reached (g, D) such that the receive r can be done in state g. By verifying that u is mb-viable as described below, we know that s is matched with r.

We check that u is mb-viable with Lemma 5.15. From Lemma 5.14 we know that u is mb-viable iff there is no unmatched send s' to q s.t. there is a $(\leq_{\mathrm{hb}} \cup <_{\mathtt{mb}})$-path from s' to s in v. We use the complement $\mathcal{D}^{co}$ of $\mathcal{D}$, which is exponential in $|\mathbb{P}|$ but can be constructed on-the-fly in linear space. We make one copy $\mathcal{D}^{co}(p')$ of $\mathcal{D}^{co}$ for every process $p' \neq q$. Each $\mathcal{D}^{co}(p')$ *tags* the first unmatched send of type $p'!q$ and s with $\bullet$. We make every $\mathcal{D}^{co}(p')$ read the tagged $\mathtt{ms}(v_1)\#\dots\#\mathtt{ms}(v_n)$ by adding the $\#$ after each atomic mb-exchange we read. Each $\mathcal{D}^{co}(p')$ should accept. This guarantees that no send of type $\overline{p'!q}$ has a $(\leq_{\mathrm{hb}} \cup <_{\mathtt{mb}})$-path to s.

For the lower bound, we use the same reduction as in Theorem 3.6, and if we reach $(\mathtt{accept})_{p\in\mathbb{P}}$, we use two other processes to do a non-mb-synchronizable gadget (see the full version of the paper [6]). This way, the CFM is mb-synchronizable if and only if the intersection of the automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ is empty. ◀

Theorem 5.16 yields two interesting corollaries. In the statements below we say that a CFM is k-mb-synchronizable if for every trace $u \in Tr_{\mathtt{mb}}(\mathcal{A})$, we have $u \equiv u_1 * \cdots * u_n$ for some mb-exchanges u_i where $|u_i| \leq k$. The next result has been shown decidable in [2] (with non-elementary complexity):

▶ **Theorem 5.17.** *Let k be an integer given in binary. The question whether a CFM is k-mb-synchronizable is* PSPACE*-complete. The lower bound already holds for k in unary.*

Proof. Using Theorem 5.16 we first check that the CFM is mb-synchronizable. Then we use the automaton $\mathcal{C}$ from Lemma 3.5 to compute pairs (g, D) of global state and set of deaf processes that are reachable by some mb-synchronous sequence. Finally we check whether the automaton of Lemma 5.10 accepts only exchanges of size at most k. Since the size of our automata is exponential the test can be done in PSPACE. The lower bound can be obtained as in the proof of Theorem 5.16 (see Figure 2). ◀

For the second result and weak synchronizability, decidability was obtained in [12]. Our proof based on automata seems more direct and simpler than the one of [12]:

▶ **Theorem 5.18.** *The question whether for a given CFM $\mathcal{A}$ there exists some k such that $\mathcal{A}$ is k-mb-synchronizable, is* PSPACE*-complete.*

Proof. For the upper bound we proceed as in the previous proof. The difference is that at the end we check whether the automaton of Lemma 5.10 accepts an infinite language from a reachable pair (g, D). The language of this automaton is infinite iff there is no k as stated by the theorem. The lower bound can be obtained as in the proof of Theorem 5.16. ◀

6 Conclusion

We have introduced a novel automata-based approach to reason about communication in the sr-round mailbox model. We showed that knowing whether a system complies with this model is PSPACE-complete. An interesting theoretical question is whether we can apply similar techniques to other types of communication. On the practical side it would be interesting to implement our algorithms and compare e.g. with existing tools like Soter [8] that targets safety properties for a relaxed model of Erlang. Our automata-based techniques may be easier to implement than previous approaches, and could even adapt to a dynamic setting.

References

1 Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996. doi:10.1006/INCO.1996.0053.

2 Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Étienne Lozes, and Amrita Suresh. A unifying framework for deciding synchronizability. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CONCUR.2021.14.

3 Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In *Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II 30*, pages 372–391. Springer, 2018.

4 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM (JACM)*, 30(2):323–342, 1983.

5 Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Comput.*, 22(1):49–71, 2009. doi:10.1007/S00446-009-0084-6.

6 Romain Delpy, Anca Muscholl, and Grégoire Sutre. An automata-based approach for synchronizable mailbox communication, 2024. arXiv:2407.06968.

7 Cinzia Di Giusto, Laetitia Laversa, and Etienne Lozes. On the k-synchronizability of systems. In *23rd International Conference on Foundations of Software Science and Computer Systems (FOSSACS 2020)*, volume 12077, pages 157–176. Springer, 2020.

8 Emanuele D'Osualdo, Jonathan Kochems, and C.-H. Luke Ong. Automatic verification of erlang-style concurrency. In Francesco Logozzo and Manuel Fähndrich, editors, *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, volume 7935 of *Lecture Notes in Computer Science*, pages 454–476. Springer, 2013. doi:10.1007/978-3-642-38856-9_24.

9 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

10 Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80(1-3):147–167, 2007.

11 Blaise Genest, Anca Muscholl, Helmut Seidl, and Marc Zeitoun. Infinite-state high-level mscs: Model-checking and realizability. *J. Comput. Syst. Sci.*, 72(4):617–647, 2006. doi:10.1016/J.JCSS.2005.09.007.

12 Cinzia Di Giusto, Laetitia Laversa, and Étienne Lozes. Guessing the buffer bound for k-synchronizability. *Int. J. Found. Comput. Sci.*, 34(8):1051–1076, 2023. doi:10.1142/S0129054122430018.

13 Loïc Hélouët and Pierre Le Maigat. Decomposition of message sequence charts. In Edel Sherratt, editor, *SAM 2000, 2nd Workshop on SDL and MSC, Col de Porte, Grenoble, France, June 26-28, 2000*, pages 47–60. Verimag, IRISA, SDL Forum, 2000.

14 Dietrich Kuske and Anca Muscholl. Communicating automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 1147–1188. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/AUTOMATA-2/9.

15 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

16 Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 350–373. Springer, 2023. doi:10.1007/978-3-031-37709-9_17.

17 Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 35:1–35:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPICS.CONCUR.2021.35`.

18 Felix Stutz. Asynchronous multiparty session type implementability is decidable - lessons learned from message sequence charts. In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPIcs*, pages 32:1–32:31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.ECOOP.2023.32`.