On the Send-Synchronizability Problem for 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

A system of communicating automata is send-synchronizable if its set of send sequences (i.e., the projection on send actions of its executions) is the same when communications are asynchronous and when they are rendez-vous synchronizations. Send-synchronizability was claimed to be decidable for the mailbox semantics (Basu and Bultan, 2011) and for the peer-to-peer semantics (Basu and Bultan, 2016). Finkel and Lozes showed in 2017 that the proofs of these results are flawed, and they proved that send-synchronizability is in fact undecidable for peer-to-peer systems. The send-synchronizability problem for mailbox systems was left open. A partial solution was recently proposed in (Di Giusto, Laversa and Peters, 2024). In this paper, we revisit the send-synchronizability problem for mailbox systems. Firstly, we show that send-synchronizability is undecidable for mailbox systems, thus closing the question left open in (Finkel and Lozes, 2023) and (Di Giusto, Laversa and Peters, 2024). Secondly, we show that send-synchronizability is decidable for the class of 1-schedulable mailbox systems. A system is 1-schedulable if every execution can be re-scheduled into an equivalent execution where each send is either immediately followed by its matching receive, or is never matched. Despite the apparent similarity between send-synchronizability and 1-schedulability, the proof that send-synchronizability is decidable for 1-schedulable mailbox systems is quite involved. We believe that the techniques that we develop in this proof could be used to address other problems on mailbox systems, such as the realizability problem.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Concurrent programming, Mailbox communication, Verification, Synchronizability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2025.15

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 mechanism for concurrent programming and distributed systems. In this model, processes running asynchronously synchronize by exchanging messages over unbounded channels. Typically, the communication follows a peer-to-peer model, which is particularly useful for reasoning about telecommunication protocols.

Recently, mailbox communication has attracted increased attention due to its role in multi-thread programming, as seen in languages like Rust or Erlang. In mailbox communication each process has a single incoming communication buffer, or mailbox, where messages from other processes are multiplexed.

Asynchronous communication poses significant challenges for formal verification, since it can easily emulate Turing machines. The quest for conditions that ensure decidability for automatic verification has led to various restrictions, such as constraints on channel behavior (e.g. lossiness [1, 12], or properties based on partial-orders [18, 14, 20]). More recently, research has focused on restricting the communication protocols themselves [6, 16, 17, 8].

While synchronous communication offers great simplicity, its relevance extends beyond verification. Many specification formalisms, such as choreography languages [2] and multi-party session types [24, 22, 23], are built upon synchronous communication. This raises a fundamental question, considering the inherent asynchrony of real-world systems: how different is an asynchronous system from its idealized synchronous version, where every sent message is instantaneously received?

Send-synchronizability offers one way to compare synchronous and asynchronous semantics. The work [13] defined a system as (send-)synchronizable if the sequence of sent messages is identical under both asynchronous and synchronous semantics. While [2] claimed decidability for mailbox systems and later extended the claim to peer-to-peer systems [3], the latter claim was refuted by [10, 11], showing undecidability for peer-to-peer systems. The decidability of send-synchronizability for mailbox systems, however, remained an open question (see, e.g., [9]), and this paper addresses it.

Send-synchronizability has been explored in several subsequent works [2, 4, 3, 11, 9] with variations such as final states, or requiring that the same state is reached in both the synchronous and asynchronous execution. For example, in [9], each process has a set of final states. This generalized version was shown to be undecidable in [9], with final states playing a key role in the proof. Our first contribution is to show that send-synchronizability remains undecidable even in the standard setting of communicating systems without final states.

Our second contribution shows the decidability of send-synchronizability for 1-schedulable systems. These systems, termed 1-synchronizable in [6] but renamed here for clarity, share similarities with the synchronous semantics. However, a key difference is that not all messages need to be received. A system is 1-schedulable if every execution can be reordered into an equivalent one where each send is either immediately followed by its corresponding receive, or remains unmatched. Being 1-schedulable is a decidable property, with PSPACE complexity [6, 16]. We show that send-synchronizability is decidable for 1-schedulable systems, but the proof is surprisingly intricated because of the unreceived messages. Our proof employs partial-order techniques (commutations), with the main challenge being the unmatched send events.

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`).

2 Definitions and Notations

2.1 Mailbox message-passing systems

We let $\mathbb{P}$ denote a finite non-empty set of *processes*, and $\mathbb{M}$ denote a finite non-empty set of *message contents*. The set of (communication) *actions* is $\mathit{Act} = \{p!q(m), q?p(m) \mid p, q \in \mathbb{P}, p \neq q, m \in \mathbb{M}\}$. 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. A communicating finite-state machine [7] 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. In the following definition, Act_p denotes the set of actions $a \in \mathit{Act}$ performed by p.

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

The usual semantics of communication is peer-to-peer, with (at most one) fifo channel for each pair of distinct processes. In this paper we focus on the *mailbox semantics*. Here, every process has a receiving fifo queue in which every other process can enqueue messages. We define the semantics by an associated global transition system.

Let $\mathcal{A} = (\mathcal{A}_p)_{p\in\mathbb{P}}$ be a CFM. The *global transition system* associated with $\mathcal{A}$ is the LTS $\mathcal{T}(\mathcal{A}) = (C_{\mathcal{A}}, Act, \to_{\mathcal{A}}, c_{in})$ with set of configurations $C_{\mathcal{A}} = G \times \prod_{p\in\mathbb{P}}(\mathbb{P}\times\mathbb{M})^*$ consisting of global states $G = \prod_{p\in\mathbb{P}} L_p$ (i.e., products of local states) and queue contents over the alphabet $\mathbb{P}\times\mathbb{M}$. Let $((\ell_p)_{p\in\mathbb{P}}, (w_p)_{p\in\mathbb{P}}) \xrightarrow{a}_{\mathcal{A}} ((\ell'_p)_{p\in\mathbb{P}}, (w'_p)_{p\in\mathbb{P}})$ 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 $w'_q = w_q\,(p,m)$ and $w'_{p'} = w_{p'}$ for $p' \neq q$.
- Receive actions: if $a = p?q(m)$ then $(q,m)\,w'_p = w_p$ and $w'_{p'} = w_{p'}$ for $p' \neq p$.

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

An *execution* of $\mathcal{T}(\mathcal{A})$ 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 2.1. Note that the queue content includes the identity of the sender. This is to exclude executions labelled by $p!q(m)\ q?r(m)$ with $p \neq r$. Without this addition such executions would be allowed in the mailbox semantics, which is clearly not intended.

▶ **Definition 2.2** (Trace). *A* trace *of a CFM $\mathcal{A}$ is a sequence $u \in Act^*$ such that there exists an initial execution of $\mathcal{T}(\mathcal{A})$ labelled by u. The set of all traces of $\mathcal{A}$ is denoted by $Tr(\mathcal{A})$.*

▶ **Definition 2.3** (Viable). *A sequence $v \in Act^*$ is called* viable *if for every $p \in \mathbb{P}$:*

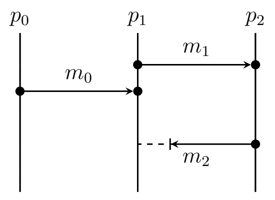
- *for every prefix u of v, the number of receives of p in u is less or equal the number of sends to p in u;*
- *for every k, if the k-th receive of p in v has label $p?q(m)$ then the k-th send to p in v has label $q!p(m)$.*

▶ **Definition 2.4** (Rendez-vous). *A sequence from Act^* is called a* rendez-vous *sequence if it is of the form $p_1!q_1(m_1)q_1?p_1(m_1)\dots p_n!q_n(m_n)q_n?p_n(m_n)$. Note that every send action is matched in a rendez-vous sequence, and that such sequences are viable. We define $Tr_{\mathtt{rdv}}(\mathcal{A}) \subseteq Tr(\mathcal{A})$ as the set of traces in $Tr(\mathcal{A})$ that are rendez-vous.*

The classical *happens-before* relation [21], frequently used in reasoning about distributed systems, orders the actions of each process and every (matched) send action before its matching receive. Because of mailbox semantics, an additional order arise between sends to the same process. The happens-before relation together with the mailbox order naturally associates a partial order with every trace, known as *message sequence chart*:

▶ **Definition 2.5** (Message Sequence Chart). *An* MSC *over $\mathbb{P}$ is an Act-labeled partially ordered set $\mathcal{M} = (E, \preceq, \lambda)$ of events E, with $\lambda : E \to Act$ and $\preceq = (\leq_{\mathbb{P}} \cup \mathrm{msg} \cup <_{\mathtt{mb}})^*$ the least partial order containing the relations $\leq_{\mathbb{P}}$,* msg *and $<_{\mathtt{mb}}$, 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.*



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

Figure 1 A sequence and its MSC. An unmatched send action is marked by a special arrowhead, as for m_2. We have here $s_0 <_{\mathtt{mb}} s_2$, so $s_0 \prec s_2$, and $s_1 \parallel s_0$, with s_i the send of message m_i.

3. *for every process p, we order two sends $e <_{\mathtt{mb}} e'$ with $e = q!p(m)$ and $e' = q'!p(m')$ if*
   - *either e is matched and e' is unmatched,*
   - *or $(e, f), (e', f') \in \mathrm{msg}$ and $f <_{\mathbb{P}} f'$.*

   We call $<_{\mathtt{mb}}$ the mailbox order*.*

For two events $e, f \in E$ we write $e \parallel f$ if neither $e \preceq f$ nor $f \preceq e$ holds. For a set of events $F \subseteq E$ we write $Past(F) = \{f \mid f \preceq e \text{ for some } e \in F\}$ and $Ftr(F) = \{f \mid e \preceq f \text{ for some } e \in F\}$ for the causal past and future of F, respectively.

If $u = u[1] \dots u[n]$ is a viable sequence of actions, then we can associate an MSC with u by setting $\mathtt{msc}(u) = (E, \preceq, \lambda)$ with $E = \{e_1, \dots, e_n\}$, $\lambda(e_i) = u[i]$, 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 $i \leq j$.
- $(e_i, e_j) \in \mathrm{msg}$ if there exists $k \geq 1$ and a process p such that $u[i]$ is the k-th send to p and $u[j]$ is the k-th receive of p.
- $e_i <_{\mathtt{mb}} e_j$ if $u[i]$ and $u[j]$ are both sends to the same process p, $u[i]$ is matched to some $u[k]$, and either $u[j]$ is unmatched, or $u[j]$ is matched to some $u[\ell]$ with $k \leq \ell$.

▶ Remark 2.6. By definition, for any viable sequence u, the associated MSC $\mathtt{msc}(u)$ is well-defined. For the converse, for every MSC $\mathcal{M}$, every (labelled) linearization of the partial order $\preceq$ of $\mathcal{M}$ is viable. Indeed, all receives of the same process are totally ordered by $\leq_{\mathbb{P}}$, and the corresponding sends are ordered in the same way because of the mailbox order. For example, the sequence shown in Figure 1 is 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.

The next definition introduces an equivalence relation $\equiv$ on sequences of actions. Two viable sequences are equivalent up to commuting adjacent actions that are not ordered by being on the same process, being a matching send/receive pair, or being two sends to the same process that are not both unmatched.

▶ **Definition 2.7** (Equivalence $\equiv$). *Two 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.*

In this paper, we define (as in [2, 10]) the *observable behavior* of a CFM as the set of projections on sends. We set $S = \{p!q(m) \mid p, q \in \mathbb{P}, p \neq q, m \in \mathbb{M}\}$. For a sequence $u \in Act^*$, we denote by $u|_S$ the projection of u on S. For any $X \subseteq Act^*$, we write $X|_S = \{u|_S \mid u \in X\}$.

▶ **Definition 2.8** (Send-synchronizability). *Let $\mathcal{A}$ be a CFM. We say that $\mathcal{A}$ is* send-synchronizable *if $Tr(\mathcal{A})|_S = Tr_{\mathtt{rdv}}(\mathcal{A})|_S$.*

The *send-synchronizability problem* is the question whether a CFM is send-synchronizable.

3 Undecidability of Send-Synchronizability for Mailbox Systems

We show in this section that the send-synchronizability problem is undecidable. We rely in our undecidability proof on the following decision problem, which is a variant of the well-known Post correspondence problem.

Pre-MPCP	
Input:	A finite sequence of pairs $(x_1, y_1), \ldots, (x_K, y_K)$, with $K \geq 1$, where $x_i, y_i \in \Sigma^*$ for some finite alphabet Σ.
Output:	Yes if there exists a sequence of indices $i_1, \ldots, i_k \in \{1, \ldots, K\}$, with $k \geq 1$, such that $i_1 = 1$, $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$ and $\lvert x_{i_1} \cdots x_{i_j} \rvert \geq \lvert y_{i_1} \cdots y_{i_j} \rvert$ for every $j \in \{1, \ldots, k\}$.

It is well-known that Pre-MPCP is undecidable [19]. The remainder of this section provides a reduction from Pre-MPCP to send-synchronizability.

Consider an instance $\mathcal{I} = (x_1, y_1), \ldots, (x_K, y_K)$ of Pre-MPCP over a finite alphabet Σ. We construct a CFM $\mathcal{A}(\mathcal{I})$ such that $\mathcal{I}$ is a positive instance of Pre-MPCP if, and only if, $\mathcal{A}(\mathcal{I})$ is not send-synchronizable. The CFM $\mathcal{A}(\mathcal{I})$ comprises three processes, namely G (the guesser), R (the relayer), and V (the verifier). So $\mathbb{P} = \{G, R, V\}$. The set of message contents is $\mathbb{M} = \Sigma \cup \overline{\Sigma} \cup \{\$, \overline{\$}, \#\}$, where $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$ is disjoint from Σ, and the messages $\$$, $\overline{\$}$ and $\#$ denote three distinct symbols that are not in $\Sigma \cup \overline{\Sigma}$. We choose to introduce a copy $\overline{\Sigma} \cup \{\overline{\$}\}$ of the alphabet $\Sigma \cup \{\$\}$ in order to easily[1] distinguish in the mailbox of V the messages that were sent by G from the messages that were sent by R. The symbols $\$$ and $\overline{\$}$ are used to mark the end of the sequences $x_{i_1} \cdots x_{i_k}$ and $\overline{y_{i_1}} \cdots \overline{y_{i_k}}$ guessed by G. It is understood that $\overline{y_1}, \ldots, \overline{y_K}$ are the words in $\overline{\Sigma}$ that are obtained from $y_1, \ldots, y_K$ by replacing each letter a by $\overline{a}$. The message $\#$ is sent by V when the two guessed sequences are equal. The LTSes $\mathcal{A}_G$, $\mathcal{A}_R$ and $\mathcal{A}_V$ of the processes G, R and V are defined in Figures 2–4. To ease understanding, the resulting communication topology, along with the potential message contents in the channels, is depicted in Figure 5. In the description of the guesser process (see Figure 2), we use the shortcuts $G!R(x_i)$ and $G!V(\overline{y_i})$ to reduce clutter. Let us explain what we mean by these shortcuts. A transition $\ell \xrightarrow{p!q(m_1 \cdots m_n)} \ell'$, where $m_i \in \mathbb{M}$, stands for the contiguous sequence of transitions $\ell = \ell_0 \xrightarrow{p!q(m_1)} \ell_1 \cdots \ell_{n-1} \xrightarrow{p!q(m_n)} \ell_n = \ell'$ where $\ell_1, \ldots, \ell_{n-1}$ are intermediate states. The observant reader will notice that the LTSes $\mathcal{A}_R$ and $\mathcal{A}_V$ contain silent transitions (from the initial state to the states D and N), which are not allowed in our definition of CFM. This is merely a notational convenience to reduce clutter in the figures. We may safely get rid of these silent transitions since send-synchronizability of a CFM is not sensitive to such transitions.

The behavior of the guesser process G should be quite clear. It sends the two guessed sequences and then waits for a message $\#$ from V, which it relays to R, signaling that the guessed sequences form a solution. The relayer process R has two operating modes, a *normal* mode (right-hand side of Figure 3) and a *dummy* mode (left-hand side of Figure 3). In the normal mode, R simply relays to V every message that it receives from G. For V to process its mailbox properly (see below), the send actions $G!V(\cdot)$ and $R!V(\cdot)$ need to be

[1] In our global transition system $\mathcal{T}(\mathcal{A})$, each message content stored in a buffer is decorated with the channel of the corresponding send action. So our construction would also work without the copy $\overline{\Sigma} \cup \{\overline{\$}\}$ of $\Sigma \cup \{\$\}$, but we believe that this copy makes our construction easier to read.

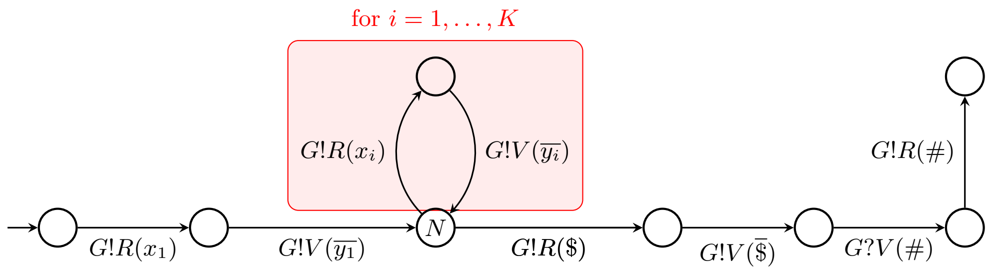


Figure 2 LTS $\mathcal{A}_G$ of the guesser process G of the CFM $\mathcal{A}(\mathcal{I})$.

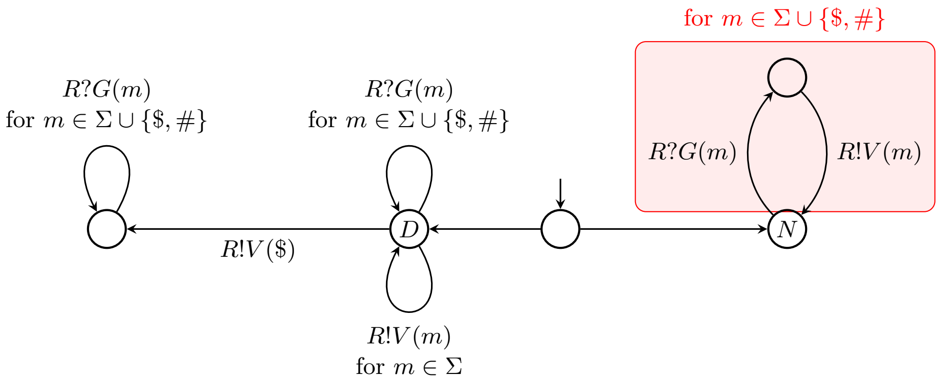

for $m \in \Sigma \cup \{\$, \#\}$
$R?G(m)$
for $m \in \Sigma \cup \{\$, \#\}$
$R?G(m)$
for $m \in \Sigma \cup \{\$, \#\}$
$R?G(m)$
$R!V(m)$
D
N
$R!V(\$)$
$R!V(m)$
for $m \in \Sigma$

Figure 3 LTS $\mathcal{A}_R$ of the relayer process R of the CFM $\mathcal{A}(\mathcal{I})$.

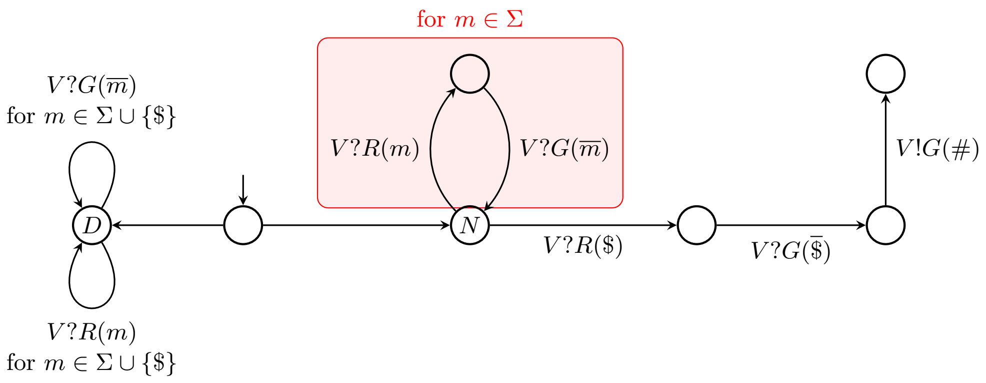


Figure 4 LTS $\mathcal{A}_V$ of the verifier process V of the CFM $\mathcal{A}(\mathcal{I})$.

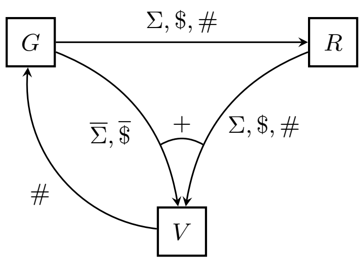


Figure 5 Topology of the CFM $\mathcal{A}(\mathcal{I})$ and message contents potentially exchanged along the channels. The $+$ sign serves as a reminder that the channels (G, V) and (R, V) are multiplexed into a single mailbox.

interleaved in lock-step (but there may be executions in $\mathcal{T}(\mathcal{A}(\mathcal{I}))$ where this does not hold). The purpose of the dummy mode is to make $\mathcal{A}(\mathcal{I})$ send-synchronizable if $\mathcal{I}$ is a negative instance of Pre-MPCP. Notice that there is no switch between modes. Analogously, the verifier process V has a *normal* mode and a *dummy* mode (right-hand and left-hand sides of Figure 4, respectively). In the normal mode, V checks that the contents of its mailbox is of the form $a_1\overline{a_1}\cdots a_n\overline{a_n}\$\overline{\$}$. It then sends $\#$ to G if that is the case (otherwise it is blocked). The purpose of the dummy mode is the same as for R. Observe that the message $\#$ sent by V to G is relayed by G to R, and then relayed by R back to V, but V never receives it.

We first show that if $\mathcal{I}$ is a positive instance of Pre-MPCP, then $\mathcal{A}(\mathcal{I})$ is not send-synchronizable.

▶ **Lemma 3.1.** *If $\mathcal{I}$ is a positive instance of Pre-MPCP then there is an initial execution in $\mathcal{T}(\mathcal{A}(\mathcal{I}))$ leading to the configuration $(N, N, N; \varepsilon, \varepsilon, \varepsilon)$.*

Proof sketch. G guesses a solution $i_1, \dots, i_k$, and sends $x_{i_1} \cdots x_{i_k}\$$ and $\overline{y_{i_1}} \cdots \overline{y_{i_k}}\overline{\$}$ to R and V, respectively. R relays $x_{i_1} \cdots x_{i_k}\$$ to V with the appropriate delay so that the mailbox of V is filled by G and R in lock-step. This is possible because our variant of PCP requires that $|x_{i_1} \cdots x_{i_j}| \geq |y_{i_1} \cdots y_{i_j}|$ for every $j \in \{1, \dots, k\}$. We end up with $a_1\overline{a_1}\cdots a_n\overline{a_n}\$\overline{\$}$ in the mailbox of V. Then V empties its mailbox by iterating the cycle on N. ◀

▶ **Corollary 3.2.** *If $\mathcal{I}$ is a positive instance of Pre-MPCP then there is a trace $u \in Tr(\mathcal{A}(\mathcal{I}))$ containing the action $R!V(\#)$, hence, $\mathcal{A}(\mathcal{I})$ is not send-synchronizable.*

Now we show that if $\mathcal{I}$ is a negative instance of Pre-MPCP, then $\mathcal{A}(\mathcal{I})$ is send-synchronizable. Some additional notations are needed. For every process $p \in \mathbb{P}$, we let $L(\mathcal{A}_p)$ denote the language recognized by the LTS $\mathcal{A}_p$ viewed as a non-deterministic automaton with every state final, and we let $u_{|p}$ denote the projection of a sequence $u \in Act^*$ on Act_p. We also define $match : S^* \to Act^*$ as the morphism given by $match(p!q(m)) = p!q(m)\, q?p(m)$.

▶ **Lemma 3.3.** *If there is a trace in $Tr(\mathcal{A}(\mathcal{I}))$ containing the action $R!V(\#)$ then $\mathcal{I}$ is a positive instance of Pre-MPCP.*

▶ **Lemma 3.4.** *For every trace $u \in Tr(\mathcal{A}(\mathcal{I}))$, if u does not contain the action $R!V(\#)$ then $match(u|_S)_{|p} \in L(\mathcal{A}_p)$ for every process $p \in \{G, R, V\}$.*

Proof sketch. Either u contains no send action to G, or u contains a single send action to G, which is $V!G(\#)$. In that case, this action is necessarily after $G!V(\overline{\$})$ and before $G!R(\#)$. In both cases, we get that $match(u|_S)_{|G} \in L(\mathcal{A}_G)$.

All sends to R are from G and all sends by R are to V. The word sent by G to R in u is a prefix of a word in $\Sigma^*\$\#$. The word sent by R to V in u is a prefix of a word in $\Sigma^*\$$, since u does not contain the action $R!V(\#)$. It follows that $match(u|_S)_{|R} \in L(\mathcal{A}_R)$, by letting R select the dummy mode.

The word sent to V in u is in $(\Sigma \cup \overline{\Sigma} \cup \{\$, \overline{\$}\})^*$, since u does not contain the action $R!V(\#)$. If u contains no send action by V, then $match(u|_S)_{|V} \in L(\mathcal{A}_V)$, by letting V select the dummy mode. Otherwise, u contains a single send action by V, which is $V!G(\#)$. The definition of $\mathcal{A}_V$ (together with the mailbox semantics) entails that the word sent to V in u is of the form $a_1\overline{a_1}\cdots a_n\overline{a_n}\$\overline{\$}$. Moreover, the action $V!G(\#)$ is necessarily after $G!V(\overline{\$})$. We derive that $match(u|_S)_{|V}$ coincides with the sequence of actions by V in u, hence, $match(u|_S)_{|V} \in L(\mathcal{A}_V)$. ◀

▶ **Corollary 3.5.** *If $\mathcal{I}$ is a negative instance of* Pre-MPCP *then $\mathcal{A}(\mathcal{I})$ is send-synchronizable.*

We then get the next theorem from Corollaries 3.2 and 3.5.

▶ **Theorem 3.6.** *The send-synchronizability problem is undecidable.*

4 Send-synchronizability and 1-schedulable Systems

In this section we show that we can decide send-synchronizability for a subclass of CFMs that we call 1-schedulable.[2] We start by defining 1-schedulable systems and a commutation relation over send sequences. Then we show that if all traces of a 1-schedulable CFM respect a certain property (and we say the CFM is *good* in this case) then we can check if the CFM is send-synchronizable. Next we show that any 1-schedulable CFM that is not good, cannot be send-synchronizable. Finally we show that we can check if a 1-schedulable CFM is good.

▶ **Definition 4.1** (1-scheduling and 1-schedulable)**.** *A viable sequence $u \in Act^*$ is a* 1-scheduling *if every send action in u is either directly followed by its receive or it is unmatched. We also say in that case that u is* 1-scheduled. *A sequence $u \in Act^*$ is* 1-schedulable *if there exists $v \equiv u$ such that v is a 1-scheduling.*

A CFM $\mathcal{A}$ is 1-schedulable if every trace $u \in Tr(\mathcal{A})$ is 1-schedulable.

From [8] (see also [16]) we know that 1-schedulability is a decidable property (PSPACE-complete). To make the characterization formal, we need some notation first. Given a binary relation $\bowtie$ over some set X, a $\bowtie$-cycle is any sequence $(x_1, \ldots, x_k) \in X$, with $k \geq 1$, such that $x_1 \bowtie x_2 \cdots \bowtie x_{k+1}$, where $x_{k+1} := x_1$. A viable sequence w is not 1-schedulable if, and only if, $\mathtt{msc}(w)$ contains a $(\prec \cup \mathrm{msg}^{-1})$-cycle with at least two distinct sends [8]. For example, the sequence $1!2(m)\, 2!1(m')\, 1?2(m')\, 2?1(m)$ is not 1-schedulable, as witnessed by the cycle $1!2(m) <_{\mathbb{P}} 1?2(m')\ \mathrm{msg}^{-1}\ 2!1(m')\ <_{\mathbb{P}} 2?1(m)\ \mathrm{msg}^{-1}\ 1!2(m)$.

Throughout this section we will assume that the CFM $\mathcal{A}$ is 1-schedulable. Our characterization of send-synchronizability will rely on partial commutations applied to the observable behaviors of rendez-vous traces.

▶ **Definition 4.2** (Commutations)**.** *We define a commutation relation $SI \subseteq S \times S$ on the alphabet S of sends as follows. Let $a = p!q(m)$ and $b = p'!q'(m')$ be two send actions. Then $(a, b) \in SI$ if $p \neq p'$, $q \neq q'$ and $q \neq p'$.*

If $(a, b) \in SI$ and $w, w' \in S^*$ then we write $wabw' \Rightarrow_{SI} wbaw'$, and we consider the reflexive-transitive closure $\overset{*}{\Rightarrow}_{SI}$ of this relation. We denote the *commutative closure* of a send sequence $v \in S^*$ as $Cl_{SI}(v) = \{v' \in S^* \mid v \overset{*}{\Rightarrow}_{SI} v'\}$. We define $Cl_{SI}(E)$, for $E \subseteq S^*$, as $\bigcup_{v \in E} Cl_{SI}(v)$.

The example in Figure 6 gives a first intuition about the role of commutations for send-synchronizability. As seen there, if a CFM has an execution w as depicted on the left, and is send-synchronizable, then it has to include also an execution as depicted on the right. This entails that if $1!0(m)2!1(m')$ is an observable behavior, then so is $2!1(m')1!0(m)$.

▶ Remark 4.3. It is readily seen that one can check if a regular language L satisfies $L = Cl_{SI}(L)$. For this it is enough to check that $wabw' \in L$ entails $wbaw' \in L$, for all words w, w' and $(a, b) \in SI$. Such a test can be done in Pspace, if L is described by an automaton.

[2] This decidable class of systems originates from [6] and it is the same as *1-synchronizable* there. To better distinguish it from send-synchronizability we decided to rename it here into 1-schedulable. Comparison with other close notions can be found in Section A.

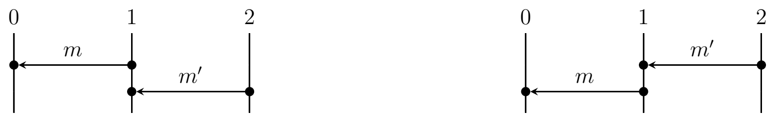


Figure 6 In the left scenario, the two send events s and s' satisfy $s \parallel s'$. Note that $(1!0(m), 2!1(m')) \in SI$, so $1!0(m)\,2!1(m') \Rightarrow_{SI} 2!1(m')\,1!0(m)$. To be send-synchronizable, a CFM must also allow the scenario on the right, where $s' \prec s$.

4.1 From 1-schedulings to arbitrary traces via commutations

In this subsection we start with a sufficient condition that ensures that the send projections of traces of $\mathcal{A}$ can be obtained via commutations from the send projections of 1-schedulings of $\mathcal{A}$ (see Proposition 4.5). The definitions that follow are used in Lemma 4.4, that states the relation between commutations and projections for a single trace.

Consider a viable sequence w. We introduce two binary relations $\dashrightarrow$ and $\ll^{w}_{\text{us}}$ over the set of events E of $\texttt{msc}(w)$, defined as follows:

- $f \dashrightarrow e$ if there exists $p \in \mathbb{P}$ such that e is an unmatched send to p, f is a send from p, and $e \parallel f$.
- $e \ll^{w}_{\text{us}} f$ if there exists $p \in \mathbb{P}$ such that e and f are unmatched sends to p, e is before f in w, and $e \parallel f$.

We say that w has a *backward* $\dashrightarrow$ *arc* if there exists two events e, f such that $f \dashrightarrow e$ and e occurs before f in w. Consider for example the trace w_1 in Figure 7a left: we have $s_0 \dashrightarrow s_1 \dashrightarrow s_2 \dashrightarrow s_0$, so w_1 has a backward $\dashrightarrow$ arc.

The objective of the $\ll^{w}_{\text{us}}$ relation is to enforce the order of unmatched sends to the same process. Given two viable sequences w and w', we write $w \equiv_{\text{us}} w'$ if $w \equiv w'$ and for every process p, the order of unmatched sends to p is the same in w and w'. Notice that for every viable sequences w, w' such that $w \equiv w'$, we have $w \equiv_{\text{us}} w'$ if, and only if, the binary relations $\ll^{w}_{\text{us}}$ and $\ll^{w'}_{\text{us}}$ coincide.

▶ **Lemma 4.4.** *Let w be a 1-scheduling and suppose that w has no backward $\dashrightarrow$ arc. Then $Cl_{SI}(w|_S) = \{w'|_S \mid w'$ is a viable sequence and $w' \equiv_{\text{us}} w\}$.*

Proof. We write $\mathcal{W} = \{w'|_S \mid w' \equiv_{\text{us}} w\}$.

We first show $Cl_{SI}(w|_S) \subseteq \mathcal{W}$ recursively on the number of commutation steps. Initially $w|_S \in \mathcal{W}$. Now we take $u \in Cl_{SI}(w|_S) \cap \mathcal{W}$, and let $u = u_0\,x\,y\,u_1$ with $x, y \in S$, $u_0, u_1 \in S^*$ and $(x, y) \in SI$. Let $u' = u_0\,y\,x\,u_1$ be the sequence of sends obtained by commuting x and y. As $u \in \mathcal{W}$, there is some $v \equiv_{\text{us}} w$ such that $v|_S = u$. We suppose w.l.o.g. that $v = v_0\,x\,y\,v_1$: if there were receives between x and y, each one could either swap with x to the left, or with y to the right, since x, y cannot be on the same process, and the sequence obtained would be $\equiv_{\text{us}}$ equivalent to v. As $(x, y) \in SI$ we know that they are not on the same process, and they do not send to the same process, so $x \parallel y$ and $v \equiv_{\text{us}} v_0\,y\,x\,v_1$, hence $u' \in \mathcal{W}$.

Now we show that $Cl_{SI}(w|_S) \supseteq \mathcal{W}$ by induction on the size of w. Initially, $\varepsilon|_S = \varepsilon$. Let $w = u\,s\,r$ be a 1-scheduling without $\dashleftarrow$ arc. Consider some $w' \equiv_{\text{us}} w$ such that $w' = x\,s\,y\,r\,z$, with $x, y, z \in Act^*$. We have $w|_S = u|_S\,s$ and $w'|_S = x|_S\,s\,(yz)|_S$. We know that $w \equiv_{\text{us}} xyz\,sr$ so $u \equiv_{\text{us}} xyz$. The case where $w = u\,s$ is dealt with similarly.

By induction, $x|_S(yz)|_S \in Cl_{SI}(u)$, so $x|_S(yz)|_S\,s \in Cl_{SI}(w)$. Now, we know that s is concurrent with every event of yz. Moreover, there is no $\dashleftarrow$ arc. So for every s' occurring in $(yz)|_S$: if $s = p!q(m)$, then s' is not on p, and it is neither a send to p (matched or not) nor to q (matched or not), so $(s', s) \in SI$, and $w'|_S \in Cl_{SI}(w)$. ◀

A viable sequence w is called *good* if there exists a 1-scheduled sequence w' such that $w \equiv_{\mathrm{us}} w'$ and w' has no backward $\dashrightarrow$ arc. Otherwise w is called *bad*. Note that every good viable sequence is necessarily 1-schedulable. A CFM $\mathcal{A}$ is good if every trace in $Tr(\mathcal{A})$ is good, and $\mathcal{A}$ is called bad otherwise.

▶ **Proposition 4.5.** *For every good CFM $\mathcal{A}$, it holds that $Tr(\mathcal{A})|_S$ is the closure under SI of $\{w \in Tr(\mathcal{A}) \mid w \text{ is a 1-scheduling}\}|_S$.*

Proof. Let K and L be defined as $K = K'|_S$ and $L = L'|_S$, where

$$K' = \{w \in Tr(\mathcal{A}) \mid w \text{ is a 1-scheduling and has no backward} \dashrightarrow \text{arc}\}$$
$$L' = \{w \in Tr(\mathcal{A}) \mid w \text{ is a 1-scheduling}\}$$

Note that $K \subseteq L \subseteq Tr(\mathcal{A})|_S$. We first show that $Tr(\mathcal{A})|_S = Cl_{SI}(K)$. Consider a trace $w \in Tr(\mathcal{A})$. As w is good, there exists a trace $w' \in Tr(\mathcal{A})$ such that $w \equiv_{\mathrm{us}} w'$, w' is a 1-scheduling and w' has no backward $\dashrightarrow$ arc. It holds that $Cl_{SI}(w'|_S) = \{u|_S \mid u \equiv_{\mathrm{us}} w'\}$ according to Lemma 4.4. As $w'|_S \in K$ and $w \equiv_{\mathrm{us}} w'$, we get that $w|_S \in Cl_{SI}(K)$. The converse inclusion $Cl_{SI}(K) \subseteq Tr(\mathcal{A})|_S$ immediately follows from Lemma 4.4. Since $K \subseteq L \subseteq Tr(\mathcal{A})|_S$ and $Tr(\mathcal{A})|_S = Cl_{SI}(K)$, we derive that $Cl_{SI}(K) \subseteq Cl_{SI}(L) \subseteq Cl_{SI}(Cl_{SI}(K)) = Cl_{SI}(K)$, hence, $Tr(\mathcal{A})|_S = Cl_{SI}(K) = Cl_{SI}(L)$. ◀

The above proposition provides our first ingredient to decide the send-synchronizability problem for 1-schedulable CFMs. Two additional ingredients are needed to complete our decidability proof. First, we will show that every bad CFM is not 1-schedulable or not send-synchronizable. Second, badness of a CFM will be shown to be decidable. We start with a characterization of bad viable sequences. For a viable sequence w, a $(\prec \cup \mathrm{msg}^{-1} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle in $\mathtt{msc}(w)$ is called *trivial* if it is of the form $(e, f, \ldots, e, f)$ with $(e, f) \in (\mathrm{msg} \cup \mathrm{msg}^{-1})$, or equivalently, if it does not contain two distinct sends.

▶ **Lemma 4.6.** *For every viable sequence w, it holds that w is bad if, and only if, there exists a $(\prec \cup \mathrm{msg}^{-1} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle in $\mathtt{msc}(w)$ that is not trivial.*

Proof sketch. Consider a viable sequence w. For short, let us write *TrvCyc* the condition that every $(\prec \cup \mathrm{msg}^{-1} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle in $\mathtt{msc}(w)$ is trivial. To prove the lemma, we show that w is good if, and only if, *TrvCyc* holds. We consider the directed graph G whose nodes are the events of $\mathtt{msc}(w)$ and whose edges are given by the relations $<_{\mathbb{P}}$, msg, $<_{\mathtt{mb}}$, msg^{-1}, $\dashrightarrow$ and $\ll^{w}_{\mathrm{us}}$. Let H denote the directed graph obtained from G by, firstly, merging events that are connected by an msg-arc, and secondly, removing all msg-arcs. It is readily seen that *TrvCyc* is equivalent to the condition that H is acyclic. If *TrvCyc* holds, then the partial order induced by H admits a linearization $x_1 \cdots x_n$. By "unmerging" events in $x_1 \cdots x_n$, we get a linearization w' of $\mathtt{msc}(w)$ such that w' is 1-scheduled and w' has no backward $\dashrightarrow$-arc nor backward $\ll^{w}_{\mathrm{us}}$-arc, hence, w is good. Conversely, if w is good then the sequence $x_1 \cdots x_n$ of nodes of H that is induced by w contains every node of H and has no backward edge, so H is acyclic, hence, *TrvCyc* holds. ◀

We now focus our attention to CFMs that are 1-schedulable. For this class of CFMs, we obtain a characterization of badness of a CFM in terms of cycles that are simpler than those of Lemma 4.6 in the sense that they contain no msg^{-1}-arc.

▶ **Lemma 4.7.** *For every 1-schedulable CFM $\mathcal{A}$, if $\mathcal{A}$ is bad then there exists a (bad) trace $w \in Tr(\mathcal{A})$ such that $\mathtt{msc}(w)$ contains a $(\prec \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle.*

Proof. Consider a bad trace $w \in \mathit{Tr}(\mathcal{A})$ of minimal length. Let Ξ denote the set of non-trivial $(<_{\mathbb{P}} \cup \mathrm{msg} \cup <_{\mathtt{mb}} \cup \mathrm{msg}^{-1} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycles in $\mathtt{msc}(w)$. We lexicographically order cycles ξ in Ξ first by number of msg^{-1}-arcs (accounting for the arc from the last event of ξ to the first event of ξ), second by length of the cycle. By Lemma 4.6, the set Ξ is non-empty. Pick a cycle ξ in Ξ of minimal length, and let us show that ξ contains no msg^{-1}-arc.

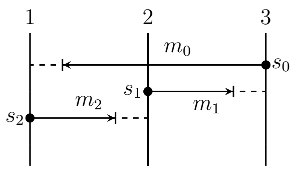
We first observe that every event of $\mathtt{msc}(w)$ is in $\mathit{Past}(C)$, where C denotes the set of events of ξ. By contradiction, assume that this is not the case. There necessarily exists an event e in $\mathtt{msc}(w)$ such that $e \notin \mathit{Past}(C)$, e is maximal for $\leq_{\mathbb{P}}$ and e is a receive or an unmatched send. Consider the sequence w' corresponding to w with e removed. It is readily seen that w' is viable, hence, $w' \in \mathit{Tr}(\mathcal{A})$. Moreover, the cycle ξ is still a non-trivial $(\prec \cup \mathrm{msg}^{-1} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle in $\mathtt{msc}(w')$. So by Lemma 4.6, w' is a bad trace in $\mathit{Tr}(\mathcal{A})$, which contradicts the minimality of w.

By contradiction, assume that ξ contains an msg^{-1}-arc. Up to a rotation of the cycle, we may assume that ξ is of the form $(e_1, \ldots, e_k)$ with $(e_k, e_1) \in \mathrm{msg}^{-1}$. Note that $k > 2$ since ξ is non-trivial. For readability, let us write $s := e_1$ and $r := e_k$. We make three observations.

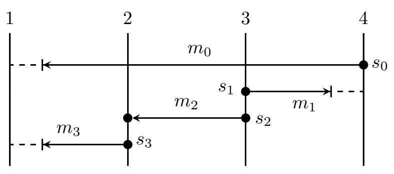
1. The event r is maximal for $\leq_{\mathbb{P}}$. By contradiction, suppose that $r <_{\mathbb{P}} f$ for some event f. Since $f \in \mathit{Past}(C)$, we get that $f \preceq e_i$ for some $1 \leq i \leq k$. Pick a $(<_{\mathbb{P}} \cup \mathrm{msg} \cup <_{\mathtt{mb}})$-path $(f_1, \ldots, f_\ell)$ with $f_1 = f$ and $f_\ell = e_i$. We get that $(e_i, \ldots, e_k, f_1, \ldots, f_{\ell-1})$ is in Ξ and has one less msg^{-1}-arc than ξ, which contradicts the minimality of ξ.
2. The event $s' := e_{k-1}$ is a send and $s' <_{\mathbb{P}} r = e_k$. Indeed, as e_k is a receive, e_{k-1} verifies $e_{k-1} <_{\mathbb{P}} r$ or $(e_{k-1}, r) \in \mathrm{msg}$. If $(e_{k-1}, r) \in \mathrm{msg}$ then $e_{k-1} = s = e_1$, hence, $(e_1, \ldots, e_{k-2})$ is in Ξ, which contradicts the minimality of ξ. So $e_{k-1} <_{\mathbb{P}} r$. It remains to show that e_{k-1} is a send. Suppose by contradiction that e_{k-1} is a receive. We either have $e_{k-2} <_{\mathbb{P}} e_{k-1}$ or $(e_{k-2}, e_{k-1}) \in \mathrm{msg}$. In the first case, $e_{k-2} <_{\mathbb{P}} r = e_k$, hence, $(e_1, \ldots, e_{k-2}, e_k)$ is in Ξ, which is impossible by minimality of ξ. In the second case, $(e_{k-2}, e_{k-1}) \in \mathrm{msg}$, hence, $e_{k-2} <_{\mathtt{mb}} s = e_1$. This entails that $(e_1, \ldots, e_{k-2})$ is in Ξ, which is impossible by minimality of ξ.
3. It holds that $s \parallel s'$. Let us prove this claim. We have $s' <_{\mathbb{P}} r$ and $(r, s) \in \mathrm{msg}^{-1}$. Note that $s \neq s'$ since s and s' are sends by distinct processes. If $s \prec s'$ then (s, s', r, s) would be a $(\prec \cup \mathrm{msg}^{-1})$-cycle with at least two distinct sends in $\mathtt{msc}(w)$, which is impossible since w is 1-schedulable. If $s' \prec s$ then $e_{k-1} = s' \prec s = e_1$, hence, $(e_1, \ldots, e_{k-1})$ is in Ξ, which contradicts the minimality of ξ.

We now derive a contradiction, which will conclude the proof that ξ contains no msg^{-1}-arc. Consider the sequence w'' corresponding to w with r removed. We have $w'' \in \mathit{Tr}(\mathcal{A})$ since r is a receive and is maximal for $\leq_{\mathbb{P}}$, by our first observation above. The event $r = e_k$ is the matching receive of $s = e_1$ in $\mathtt{msc}(w)$, so s becomes an unmatched send in $\mathtt{msc}(w'')$. In $\mathtt{msc}(w'')$, the event $s' = e_{k-1}$ is a send and s is a send to the process performing s', by our second observation above. Moreover, as $s \parallel s'$ in $\mathtt{msc}(w)$, by our third observation above, we still have $s \parallel s'$ in $\mathtt{msc}(w'')$. It follows that $s' \dashrightarrow s$ in $\mathtt{msc}(w'')$, hence, $e_{k-1} \dashrightarrow e_1$. We derive that $(e_1, \ldots, e_{k-1})$ is a non-trivial $(\prec \cup \mathrm{msg}^{-1} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle in $\mathtt{msc}(w'')$. This entails, by Lemma 4.6, that w'' is bad, which contradicts the minimality of w. ◀

▶ Remark 4.8. Given a viable sequence w, a $(\prec \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle $(e_1, \ldots, e_k)$ in $\mathtt{msc}(w)$ is called *genuine* if $e_1, \ldots, e_k$ are pairwise-distinct and, for every $1 \leq i < j \leq k$, we have $e_i \prec e_j$ implies $j = i + 1$ and $e_j \prec e_i$ implies $i = 1$ and $j = k$. Informally, a genuine cycle is a simple cycle such that for every events e, f on the cycle, if $e \prec f$ then the subpath of the cycle from e to f is a single $\prec$-arc. It is routinely checked that if $\mathtt{msc}(w)$ contains a $(\prec \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle then it contains a genuine $(\prec \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle.



(a) MSC of sequence $w_1 = 3!1(m_0)\,2!3(m_1)\,1!2(m_2)$.



(b) MSC of sequence $w_2 = 4!1(m_0)\,3!4(m_1)\,3!2(m_2)\,2?3(m_2)\,2!1(m_3)$.

Figure 7 Sequences with pattern, and their MSC.

4.2 Bad CFMs

Our goal in this section is to analyze bad, 1-schedulable CFMs. We will show a necessary condition for badness in terms of patterns (Proposition 4.10), and we will show that the presence of a pattern prevents a CFM to be send-synchronizable (Proposition 4.11).

We focus on 1-scheduled traces, as they can be effectively constructed from the CFM using automata. The binary relation $\ll^w_{\text{us}}$ needs to be relaxed as we might need to re-order unmatched sends of a 1-scheduling in $Tr(\mathcal{A})$ in order to exhibit a bad 1-schedulable trace in $Tr(\mathcal{A})$. Given a viable sequence w, we introduce the binary relation $\|_{\text{us}}$ over the set of events E of $\texttt{msc}(w)$, defined by $e \|_{\text{us}} f$ if there exists $p \in \mathbb{P}$ such that e and f are unmatched sends to p and $e \| f$. Note that $\|_{\text{us}}$ is symmetric and contains $\ll^w_{\text{us}}$.

A *pattern* for a viable sequence w is an alternating sequence of send events $(x_1,y_1,\ldots,x_k,y_k)$ of $\texttt{msc}(w)$ satisfying, with the convention that $x_{k+1} := x_1$, the following conditions:

1. $x_i \preceq y_i$, for every $1 \le i \le k$,
2. $y_i \dashrightarrow x_{i+1}$ or $y_i \|_{\text{us}} x_{i+1}$, for every $1 \le i \le k$,
3. $y_i \dashrightarrow x_{i+1}$ for some $1 \le i \le k$,
4. $x_i \| x_j$ and $y_i \| y_j$ and $x_i \| y_j$, for every $1 \le i, j \le k$ with $i \neq j$.

▶ **Example 4.9.** Figure 7 shows two examples of patterns. In Figure 7a the sequence w_1 has the pattern $(s_0, s_0, s_1, s_1, s_2, s_2)$, with $s_0 \dashrightarrow s_1$, $s_1 \dashrightarrow s_2$ and $s_2 \dashrightarrow s_0$.
The pattern in Figure 7b is (s_0, s_0, s_1, s_3), with $s_0 \dashrightarrow s_1$, $s_1 \prec s_3$, and $s_3 \|_{\text{us}} s_0$.

▶ **Proposition 4.10.** *Let $\mathcal{A}$ be 1-schedulable CFM. If $\mathcal{A}$ is bad then there exists a trace $w \in Tr(\mathcal{A})$ that admits a pattern $(x_1, y_1, \ldots, x_k, y_k)$.*

Proof. By Lemma 4.7 and Remark 4.8, there exists a trace $w \in Tr(\mathcal{A})$ such that $\texttt{msc}(w)$ contains a genuine $(\prec \cup \dashrightarrow \cup \ll^w_{\text{us}})$-cycle ξ. Observe that ξ necessarily contains a $\dashrightarrow$ arc. The reason is that $\prec$ and $\ll^w_{\text{us}}$ are both contained in the total order over positions of w, hence, there is no $(\prec \cup \ll^w_{\text{us}})$-cycle in $\texttt{msc}(w)$. In order to transform $\xi = (e_1, \ldots, e_k)$ into a pattern, we insert "identity" arcs of the form (e_i, e_i) as needed to get an alternating sequence $(x_1, y_1, \ldots, x_k, y_k)$ satisfying the first three conditions of the definition of patterns (recall that $\ll^w_{\text{us}}$ is contained in $\|_{\text{us}}$). The fourth (and last) condition holds because ξ is genuine. ◀

▶ **Proposition 4.11.** *Let $\mathcal{A}$ be 1-schedulable. If some 1-scheduling $w \in Tr(\mathcal{A})$ contains a pattern, then $\mathcal{A}$ is not send-synchronizable.*

For the proof of Proposition 4.11 we first provide two technical lemmas. Recall that patterns are cycles. Proposition 4.11 will show a contradiction to send-synchronizability by listing the cycle in two ways. Lemma 4.12 identifies traces corresponding to different listings of the cycle in a such way that the part before the cycle is the same. Lemma 4.13 relates arcs of two equivalent traces with the same send-projection, one of which is rendez-vous.

▶ **Lemma 4.12.** *Let $w \in Tr(\mathcal{A})$ contain some pattern $(x_1, y_1, \dots, x_k, y_k)$. Let also $Z = \{x_1, y_1, \dots, x_k, y_k\}$. Then for every $1 \le i \le k$ there exist $u, v, u' \in Act^*$ with $uvu' \equiv w$ satisfying the following:*

- *u is a linearization of $Past(Z) \setminus Ftr(Z)$*
- *uv is a linearization of $Past(Z)$*
- *v contains $x_i, y_i, \dots, x_k, y_k, x_1, y_1, \dots x_{i-1}, y_{i-1}$ as subsequence.*

Proof. As in Lemma 4.7 we start with a prefix containing all events that are ordered causally before the events of the pattern: u is a linearization of $Past(Z) \setminus Ftr(Z)$.

Consider some $j, \ell \in \{1, \dots, k\}$, $j \ne \ell$. For readability we write Z_i for the infix $Past(y_i) \cap Ftr(x_i)$ of $\texttt{msc}(w)$, for every i.

Let also $e \in Z_j$ and $f \in Z_\ell$. If $e \preceq f$ would hold true then, using $f \preceq y_\ell$ and $x_j \preceq e$, we would also get $x_j \preceq y_\ell$, contradicting the definition of pattern. By symmetry, we conclude that $e \parallel f$ for all such e, f.

Therefore, for $1 \le i \le k$ we can define v as linearization of $Z_i, Z_{i+1}, \dots Z_k, Z_1, \dots Z_{i-1}$. By definition, v contains the subsequence $x_i, y_i, \dots, x_k, y_k, x_1, y_1, \dots x_{i-1}, y_{i-1}$. ◀

▶ **Lemma 4.13.** *Consider $w \in Tr(\mathcal{A})$ and $w' \in Tr_{\texttt{rdv}}(\mathcal{A})$ with $w|_S = w'|_S$. Then for every send events $x \ne y$ in w, with x before y in w:*

- *If $x \dashrightarrow y$ in w, then $x <_{\mathbb{P}} z$, where z is the matching receive of y in w': $(y, z) \in \text{msg}$.*
- *If $x \parallel_{\text{us}} y$ in w, then $x <_{\texttt{mb}} y$ in w'.*
- *If $x \prec y$ in w then $x \prec y$ in w' by the same causal path.*

Proof. If $x \dashrightarrow y$ in w, we know that x is a send on some process p and y is an unmatched send to process p. Because w' is a rendez-vous execution, with x before y, we get $x <_{\mathbb{P}} z$.

Now suppose that $x \parallel_{\text{us}} y$ in w. Then x and y are two unmatched sends to the same process. The two sends are matched in w', with $x' <_{\mathbb{P}} y'$, where $(x, x') \in \text{msg}$, $(y, y') \in \text{msg}$. Thus $x \prec y$ in w' because of $<_{\texttt{mb}}$.

Finally, if $x \prec y$ in w, then $x \prec y$ holds also in w' because every arc of type $<_{\mathbb{P}}, \text{msg}, <_{\texttt{mb}}$ on the causal path from x to y is preserved in w'. ◀

Proof of Proposition 4.11. We assume that $\mathcal{A}$ is send-synchronizable, and consider some $w \in Tr(\mathcal{A})$ with pattern $(x_1, y_1, \dots, x_k, y_k)$. We will assume that the pattern is minimal w.r.t. the following parameter: the size of the pattern is the sum of the lengths of the shortest paths from x_j to y_j, over all j.

By Lemma 4.12 we can assume that $w = uv$ is a linearization of $Past(Z)$, with $Z = \{x_1, y_1, \dots, x_k, y_k\}$, and $x_1, y_1, \dots, x_k, y_k$ occurs as subsequence of v. In addition, u is linearization of $Past(Z) \setminus Ftr(Z)$.

We know that $(x_1, y_1, \dots, x_k, y_k)$ is a pattern in v, and we can assume w.l.o.g. that $y_k \dashrightarrow x_1$. Note that $k > 1$ since $x_1 \preceq y_1$ (by definition of patterns) and $x_1 \parallel y_k$ (by definition of $\dashrightarrow$). By Lemma 4.12, for every $1 \le i \le k$ there exists some $v' \equiv v$ such that $x_{i+1}, y_{i+1}, \dots, x_k, y_k, x_1, y_1, \dots, x_i, y_i$ is a subsequence of v'. Chose for example $i = 1$.

As $\mathcal{A}$ is send-synchronizable, there exist $z, z' \in Tr_{\texttt{rdv}}(\mathcal{A})$ such that $(uv)|_S = z|_S$ and $(uv')|_S = z'|_S$. From Lemma 4.13, we get that there exists a $(\prec \cup \text{msg}^{-1})^*$-path from x_1 to y_k in z not going through $rcv(x_1)$, the receive matching x_1. Indeed, for each $1 \le i \le k$, we have $x_i \prec y_i$ in w, so we still have it in z and it does not go through $rcv(x_1)$. We also have $y_i \dashrightarrow x_{i+1}$ or $y_i \parallel_{\text{us}} x_{i+1}$ in w, for every i. If $y_i \dashrightarrow x_{i+1}$ then $y_i <_{\mathbb{P}} rcv(x_{i+1}) \, \text{msg}^{-1} \, x_{i+1}$ in z. If $y_i \parallel_{\text{us}} x_{i+1}$, then $y_i <_{\texttt{mb}} x_{i+1}$ in z. In both cases the arcs do not use $rcv(x_1)$.

We inspect now $M := \texttt{msc}(z)$. Let p be the process of the send y_k (recall that it is also the process of $rcv(x_1)$, because $y_k \dashrightarrow x_1$). Because of pattern minimality, we can show that the only send event *on* p in v is y_k. Suppose that some $s \ne y_k$ is a send on process p with

s in v. By definition of u, v, we have $x_k \preceq s \prec y_k$. But then the pattern $(x_1, y_1, \ldots, x_k, s)$ would be smaller, contradiction. Again by minimality, we can show that the only send *to* p in v is x_1. First there is no matched send to p in v, because such a send s would satisfy $s \prec x_1$ as x_1 is an unmatched send to p, hence, if $x_j \preceq s \preceq y_j$ with $1 \leq j \leq k$, then $x_j \prec x_1$, which is forbidden by the definition of patterns. Now if we had some $s \neq x_1$ unmatched send event to p, with say $x_j \preceq s \preceq y_j$ and $1 \leq j < k$, then the pattern $(s, y_j, \ldots, x_k, y_k)$ would be smaller. Finally, if $j = k$, so $x_k \preceq s \prec y_k$, then the predecessor of s on the causal path from x_k to s is an event on p, and we can shorten path from x_k to y_k, obtaining a smaller pattern.

Let us construct $z^\sharp$ from z by moving $rcv(x_1)$ at the end of the sequence. Observe that $z = match(z|_S) = match(u|_S)match(v|_S)$ and similarly $z' = match(u|_S)match(v'|_S)$, where $match : S^* \to Act^*$ is the morphism defined by $match(p!q(m)) = p!q(m)\, q?p(m)$. We derive from the previous paragraph that the sequence of actions by p is $rcv(x_1)y_k$ in $match(v|_S)$ and $y_k rcv(x_1)$ in $match(v'|_S)$. As $rcv(x_1)$ is the last receive by p in z, moving it at the end of the sequence preserves viability, so $z^\sharp$ is viable. Moreover, the sequence of actions by p in $z^\sharp$ coincides with that of z', and the sequence of actions by $q \neq p$ in $z^\sharp$ coincides with that of z. Since $z \in Tr(\mathcal{A})$ and $z' \in Tr(\mathcal{A})$, we get that $z^\sharp \in Tr(\mathcal{A})$.

However, note that now there is a $(\prec \cup \mathrm{msg}^{-1})$-cycle with at least two distinct sends in $M^\sharp := \mathtt{msc}(z^\sharp)$. Indeed, we still have in $M^\sharp$ the $(\prec \cup \mathrm{msg}^{-1})^+$ -path of M from x_1 to y_k, as this path does not go through $rcv(x_1)$, and $y_k <_{\mathbb{P}} rcv(x_1)$ in $M^\sharp$, finishing the cycle with $(rcv(x_1), x_1) \in \mathrm{msg}^{-1}$. So $z^\sharp$ is not 1-schedulable because it contains a non-trivial $(\prec \cup \mathrm{msg}^{-1})^+$-cycle with at least two sends, which contradicts our assumption that $\mathcal{A}$ is 1-schedulable. ◀

4.3 Wrap-Up

In this section, we prove that send-synchronizability is decidable for 1-schedulable CFMs, by assembling the results from the previous subsections. We start with the following observation, which is an immediate consequence of Propositions 4.5, 4.10, and 4.11.

▶ **Proposition 4.14.** *For every 1-schedulable CFM $\mathcal{A}$, it holds that $\mathcal{A}$ is send-synchronizable if, and only if, (1) $\mathcal{A}$ is good, (2) $Tr_{\mathtt{rdv}}(\mathcal{A})|_S = \{w \in Tr(\mathcal{A}) \mid w \text{ is a } 1\text{-scheduling}\}|_S$ and (3) $Tr_{\mathtt{rdv}}(\mathcal{A})|_S$ is closed under SI.*

▶ **Proposition 4.15.** *It is decidable to check if a 1-schedulable CFM is good.*

Proof sketch. Let $\mathcal{A}$ be a 1-schedulable CFM. From Proposition 4.10, we know that $\mathcal{A}$ is bad if there exists a trace with a pattern. The existence of a pattern can be checked on a 1-scheduling, as by construction two equivalent traces have the same patterns, and $\mathcal{A}$ is 1-schedulable. Moreover the set of 1-scheduling of $\mathcal{A}$ is regular.

We guess a bad 1-scheduling on-the-fly, together with the sends that form the pattern. We can assume that the pattern occurs in the order of the sequence. If there exists some bad trace in $Tr(\mathcal{A})$, then there is one with a pattern of size at most $2 \times |\mathbb{P}|$, as otherwise there would be at least three sends of the pattern on the same process, and we would have a shortcut.

Let $(x_1, y_1, \ldots, x_k, y_k)$ be the pattern we read. We need to check that

- for each $0 \leq i \leq k$, $x_i \preceq y_i$ and $y_i \parallel x_{i+1}$ with either $\dashrightarrow$ or $\parallel_{\mathrm{us}}$, and at least one $\dashrightarrow$-arc.
- and for each $0 \leq i, j \leq k$ with $i \neq j$, $x_i \parallel x_j$ and $y_i \parallel y_j$ and $x_i \parallel y_j$.

To do so, we will keep for each send s of the pattern the set of processes with actions causally ordered after s, and we will update these sets while reading the bad 1-scheduling. So we need a memory of size $O(|\mathbb{P}|^2)$. ◀

From the two previous propositions, we get the following theorem.

▶ **Theorem 4.16.** *The question whether a 1-schedulable CFM is send-synchronizable is* PSPACE-*complete.*

Proof. Let $\mathcal{A}$ be a 1-schedulable CFM. From Proposition 4.15, we know that we can check in polynomial space if $\mathcal{A}$ is good. The two sets $Tr_{\texttt{rdv}}(\mathcal{A})$ and $\{w \in Tr(\mathcal{A}) \mid w \text{ is a 1-scheduling}\}$ are regular, so we can check in PSPACE if they are equal. And from Remark 4.3, we know that we can check if $Tr_{\texttt{rdv}}(\mathcal{A})|_S$ is closed under SI in PSPACE.

For PSPACE-hardness, we reduce from the automata intersection problem as in [8], with a send that cannot be received if the intersection of the automata is not empty. ◀

5 Conclusion

We showed that send-synchronizability for mailbox CFMs is undecidable, thus closing the question left open in [11, 9]. We also showed that we can decide in PSPACE if a 1-schedulable CFM (which is a property that can be checked in PSPACE) is send-synchronizable. We think that the techniques developed for the proof could be used to answer others problems for mailbox CFMs, like the realizability problem. Moreover, even if the class of 1-schedulable systems is decidable, it is quite restrictive. It can be interesting to see if our techniques can be used for k-schedulable CFMs and the more general class of CFMs studied in [8].

References

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

2 Samik Basu and Tevfik Bultan. Choreography conformance via synchronizability. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 795–804. ACM, 2011. doi:10.1145/1963405.1963516.

3 Samik Basu and Tevfik Bultan. On deciding synchronizability for asynchronously communicating systems. *Theoretical Computer Science*, 656:60–75, 2016. doi:10.1016/J.TCS.2016.09.023.

4 Samik Basu, Tevfik Bultan, and Meriem Ouederni. Synchronizability for verification of asynchronously communicating systems. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, volume 7148 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 2012. doi:10.1007/978-3-642-27940-9_5.

5 Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Étienne Lozes, and Amrita Suresh. A unifying framework for deciding synchronizability. In *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.

6 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*, volume 10982 of *Lecture Notes in Computer Science*, pages 372–391. Springer, 2018. doi:10.1007/978-3-319-96142-2_23.

7 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

8 Romain Delpy, Anca Muscholl, and Grégoire Sutre. An automata-based approach for synchronizable mailbox communication. In *35th International Conference on Concurrency Theory, CONCUR 2024, September 9-13, 2024, Calgary, Canada*, volume 311 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPICS.CONCUR.2024.22`.

9 Cinzia Di Giusto, Laetitia Laversa, and Kirstin Peters. Synchronisability in mailbox communication. In Proceedings Combined 31st International Workshop on*Theoretical Computer Science Expressiveness in Concurrency*and 21st Workshop on *Structural Operational Semantics,* Calgary, Canada, 9th September 2024, volume 412 of *Electronic Proceedings in Theoretical Computer Science*, pages 19–34. Open Publishing Association, 2024. `doi:10.4204/EPTCS.412.3`.

10 Alain Finkel and Étienne Lozes. Synchronizability of communicating finite state machines is not decidable. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 122:1–122:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPICS.ICALP.2017.122`.

11 Alain Finkel and Étienne Lozes. Synchronizability of communicating finite state machines is not decidable. *Logical Methods in Computer Science*, 19(4), 2023. `doi:10.46298/LMCS-19(4:33)2023`.

12 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001. `doi:10.1016/S0304-3975(00)00102-X`.

13 Xiang Fu, Tevfik Bultan, and Jianwen Su. Synchronizability of conversations among web services. *IEEE Transactions on Software Engineering*, 31(12):1042–1055, 2005. `doi:10.1109/TSE.2005.141`.

14 Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80(1-3):147–167, 2007. URL: `http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-09`.

15 Cinzia Di Giusto, Davide Ferré, Laetitia Laversa, and Étienne Lozes. A partial order view of message-passing communication models. *Proceedings of the ACM on Programming Languages*, 7(POPL):1601–1627, 2023. `doi:10.1145/3571248`.

16 Cinzia Di Giusto, Laetitia Laversa, and Étienne Lozes. On the k-synchronizability of systems. In *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2020. `doi:10.1007/978-3-030-45231-5_9`.

17 Cinzia Di Giusto, Laetitia Laversa, and Étienne Lozes. Guessing the buffer bound for k-synchronizability. *International Journal of Foundations of Computer Science*, 34(8):1051–1076, 2023. `doi:10.1142/S0129054122430018`.

18 Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, Milind A. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, 2005. `doi:10.1016/J.IC.2004.08.004`.

19 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

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

21 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. `doi:10.1145/359545.359563`.

22 Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. Complete multiparty session type projection with automata. In *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.

23 Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. In *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.

24 Felix Stutz. Asynchronous multiparty session type implementability is decidable - lessons learned from message sequence charts. In *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.

A 1-schedulability and related notions

Similar notions to 1-schedulability can be found in several papers [5, 17]. The way unmatched sends are treated in the literature may differ in a significant way. Our notion of 1-schedulable is the same as strongly 1-synchronisable in [5]. The notion of RSC in [15] is more restrictive than 1-schedulable, as it does not allow to permute two unmatched sends to the same process.

Take as an example the trace 0!2(a) 0!1(b) 2!1(c) 2?0(a). It is 1-schedulable since it is equivalent to 2!1(c) 0!2(a) 2?0(a) 0!1(b). But it is not RSC according to [15], since RSC disallows permuting 0!1(b), 2!1(c). Note however that every 1-scheduling is an RSC execution.

B A necessary and sufficient condition for send-synchronizability

Let $\mathcal{A}$ be a CFM. Recall that $\mathcal{A}$ is send-synchronizable if, and only if, for every trace $u \in Tr(\mathcal{A})$, there exists a trace $v \in Tr_{\mathtt{rdv}}(\mathcal{A})$ with $u|_S = v|_S$. In fact, the trace v is unique if it exists. Indeed, the only candidate for v is $match(u|_S)$. Recall that $match : S^* \to Act^*$ is the morphism defined by $match(p!q(m)) = p!q(m)\, q?p(m)$. Observe that $match(u)|_S = u$ for all $u \in S^*$. This leads us to the characterization of send-synchronizability in Lemma B.1 below. The following notations are used in this lemma. For every process $p \in \mathbb{P}$, $L(\mathcal{A}_p)$ is the language of the LTS $\mathcal{A}_p$ (viewed as a non-deterministic automaton with every state final) and $u_{|p}$ is the projection of $u \in Act^*$ on Act_p.

▶ **Lemma B.1.** *A CFM $\mathcal{A}$ is send-synchronizable if, and only if, for every trace $u \in Tr(\mathcal{A})$, the sequence $v = match(u|_S)$ satisfies $v_{|p} \in L(\mathcal{A}_p)$ for every process $p \in \mathbb{P}$.*

Proof. Assume that $\mathcal{A}$ is send-synchronizable. Let $u \in Tr(\mathcal{A})$ and define $v = match(u|_S)$. Since $\mathcal{A}$ is send-synchronizable, there exists $u' \in Tr_{\mathtt{rdv}}(\mathcal{A})$ such that $u|_S = u'|_S$. We have $match(u'|_S) = u'$ since $u' \in Tr_{\mathtt{rdv}}(\mathcal{A})$. It follows that $u' = v$, hence, $v \in Tr_{\mathtt{rdv}}(\mathcal{A})$. This entails that $v_{|p} \in L(\mathcal{A}_p)$ for every process $p \in \mathbb{P}$.

Conversely, let $u \in Tr(\mathcal{A})$ and assume that $v = match(u|_S)$ satisfies $v_{|p} \in L(\mathcal{A}_p)$ for every process $p \in \mathbb{P}$. The sequence v is clearly rendez-vous, according to the definition of *match*. We derive that $v \in Tr_{\mathtt{rdv}}(\mathcal{A})$, since $v_{|p} \in L(\mathcal{A}_p)$ for every process $p \in \mathbb{P}$. The observation that $v|_S = match(u|_S)|_S = u|_S$ entails that $u|_S \in Tr_{\mathtt{rdv}}(\mathcal{A})|_S$, which concludes the proof. ◀

C Proofs of Section 3

▶ **Lemma 3.1.** *If $\mathcal{I}$ is a positive instance of* Pre-MPCP *then there is an initial execution in $\mathcal{T}(\mathcal{A}(\mathcal{I}))$ leading to the configuration $(N, N, N; \varepsilon, \varepsilon, \varepsilon)$.*

Proof. Consider a sequence of indices $i_1, \dots, i_k \in \{1, \dots, K\}$, with $k \geq 1$, such that $i_1 = 1$, $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$ and $|x_{i_1} \cdots x_{i_j}| \geq |y_{i_1} \cdots y_{i_j}|$ for every $j \in \{1, \dots, k\}$. Let us write $a_1 \cdots a_n$, with $a_i \in \Sigma$, the word $x_{i_1} \cdots x_{i_k}$ (or, equivalently, the word $y_{i_1} \cdots y_{i_k}$). Define $e_j = |x_{i_1} \cdots x_{i_j}|$ and $f_j = |y_{i_1} \cdots y_{i_j}|$, for every $j \in \{1, \dots, k\}$. We also put $e_0 = f_0 = 0$. Our assumptions on $i_1, \dots, i_k$ entail, firstly, that $e_k = f_k = n$, and, secondly, that $e_j \geq f_j$, $x_{i_j} = a_{e_{j-1}+1} \cdots a_{e_j}$ and $y_{i_j} = a_{f_{j-1}+1} \cdots a_{f_j}$, for all $j \in \{1, \dots, k\}$.

We show that for every $j \in \{1, \dots, k\}$, there is an execution ρ_j in $\mathcal{T}(\mathcal{A}(\mathcal{I}))$ from the configuration $(N, N, N; \varepsilon, v_j^R, v_j^V)$, with $v_j^R = a_{f_{j-1}+1} \cdots a_{e_{j-1}}$ and $v_j^V = a_1 \overline{a_1} \cdots a_{f_{j-1}} \overline{a_{f_{j-1}}}$, to the configuration $(N, N, N; \varepsilon, w_j^R, w_j^V)$, with $w_j^R = a_{f_j+1} \cdots a_{e_j}$ and $w_j^V = a_1 \overline{a_1} \cdots a_{f_j} \overline{a_{f_j}}$. In the execution ρ_j, the sequence of actions performed by G is $G!R(x_{i_j})\, G!V(\overline{y_{i_j}})$, the sequences of receive and send actions performed by R are $R?G(y_{i_j})$ and $R!V(y_{i_j})$, respectively, and V does not move. These sequences of actions are permitted by the LTSes $\mathcal{A}_G$ and $\mathcal{A}_R$ (see Figures 2 and 3). To make it clear that such an execution ρ_j exists, we need to show that these actions can be interleaved in a way that produces an viable sequence (that is, in addition, permitted by the LTS $\mathcal{A}_R$). To reduce clutter, we define $b_1 = a_{f_{j-1}+1}, \dots, b_m = a_{f_j}$, where $m = f_j - f_{j-1}$. Note that $y_{i_j} = b_1 \cdots b_m$. Formally, the viable sequence labeling the execution ρ_j is

$$G!R(x_{i_j}) \cdot R?G(b_1)\, R!V(b_1)\, G!V(\overline{b_1}) \cdots R?G(b_m)\, R!V(b_m)\, G!V(\overline{b_m})$$

This sequence is viable because the contents z of R's mailbox after $G!R(x_{i_j})$ verifies $z = v_j^R \cdot x_{i_j} = a_{f_{j-1}+1} \cdots a_{e_j}$, hence, z admits $b_1 \cdots b_m$ as prefix, since $b_1 \cdots b_m = a_{f_{j-1}+1} \cdots a_{f_j}$ and $e_j \geq f_j$. It is readily seen that the configuration $(N, N, N; \varepsilon, w_j^R, w_j^V)$ obtained after ρ_j verifies $w_j^R = a_{f_j+1} \cdots a_{e_j}$ and $w_j^V = a_1 \overline{a_1} \cdots a_{f_j} \overline{a_{f_j}}$.

We also observe that there is an execution ρ_{k+1} in $\mathcal{T}(\mathcal{A}(\mathcal{I}))$ from the configuration $(N, N, N; \varepsilon, \varepsilon, v_k^V)$, with $v_k^V = a_1 \overline{a_1} \cdots a_n \overline{a_n}$, to the configuration $(N, N, N; \varepsilon, \varepsilon, \varepsilon)$. The trace of ρ_{k+1} is the viable sequence $V?R(a_1)\, V?G(\overline{a_1}) \cdots V?R(a_n)\, V?G(\overline{a_n})$. This sequence of actions is permitted by the LTS $\mathcal{A}_V$ (see Figure 4).

By concatenating the executions $\rho_1, \dots, \rho_{k+1}$, we get an execution from $(N, N, N; \varepsilon, \varepsilon, \varepsilon)$ to $(N, N, N; \varepsilon, \varepsilon, \varepsilon)$. But this execution does not start from the initial configuration of $\mathcal{T}(\mathcal{A}(\mathcal{I}))$. Along the execution ρ_1, G does $G!R(x_1)G!V(\overline{y_1})$, since $i_1 = 1$. So G could as well have started from its initial state. Obviously, the same goes for R and V, since they can move silently from their initial state to the state N. Hence, there exists an execution ρ_1' from the initial configuration of $\mathcal{T}(\mathcal{A}(\mathcal{I}))$ to the configuration obtained after ρ_1. The concatenation $\rho_1' \rho_2 \cdots \rho_K$ provides an initial execution leading to the configuration $(N, N, N; \varepsilon, \varepsilon, \varepsilon)$. ◀

▶ **Corollary 3.2.** *If $\mathcal{I}$ is a positive instance of* Pre-MPCP *then there is a trace $u \in Tr(\mathcal{A}(\mathcal{I}))$ containing the action $R!V(\#)$, hence, $\mathcal{A}(\mathcal{I})$ is not send-synchronizable.*

Proof. Starting from the configuration $(N, N, N; \varepsilon, \varepsilon, \varepsilon)$, there is an execution in $\mathcal{T}(\mathcal{A}(\mathcal{I}))$ labelled by

$$G!R(\$)\, R?G(\$)\, R!V(\$)\, G!V(\overline{\$})\, V?R(\$)\, V?G(\overline{\$})\, V!G(\#)\, G?V(\#)\, G!R(\#)\, R?G(\#)\, R!V(\#)\,.$$

If $\mathcal{I}$ is a positive instance of Pre-MPCP then, according to Lemma 3.1 and the above observation, there is a trace $u \in Tr(\mathcal{A}(\mathcal{I}))$ ending with $R!V(\#)$. Obviously, it holds that $u|_S \in Tr(\mathcal{A}(\mathcal{I}))|_S$. But, as V contains no $V?R(\#)$ transition, no trace in $Tr_{\texttt{rdv}}(\mathcal{A}(\mathcal{I}))$ contains the action $R!V(\#)$. Hence, $u|_S \notin Tr_{\texttt{rdv}}(\mathcal{A}(\mathcal{I}))|_S$, which entails that $\mathcal{A}(\mathcal{I})$ is not send-synchronizable. ◀

▶ **Lemma 3.3.** *If there is a trace in $Tr(\mathcal{A}(\mathcal{I}))$ containing the action $R!V(\#)$ then $\mathcal{I}$ is a positive instance of* Pre-*MPCP.*

Proof. Consider an initial execution ρ in $\mathcal{T}(\mathcal{A}(\mathcal{I}))$ ending with the action $R!V(\#)$. By construction, ρ also necessarily contains $R?G(\#)$, hence, it also contains $G!R(\#)$ and $G?V(\#)$, which entails that it contains $V!G(\#)$. Therefore, in the execution ρ, the processes R and V have each selected the normal mode, since there is no $R!V(\#)$ transition nor $V!G(\#)$ transition in the dummy modes. Let π denote the prefix of ρ that ends just before the $V!G(\#)$ action, and let u denote the trace labelling π. According to the definition of the LTSes $\mathcal{A}_G$, $\mathcal{A}_R$ and $\mathcal{A}_V$ (see Figures 2–4), the projections $u_{|G}$, $u_{|R}$ and $u_{|V}$ are as follows:

$$\begin{aligned}
u_{|G} &= G!R(x_{i_1})\, G!V(\overline{y_{i_1}}) \cdots G!R(x_{i_k})\, G!V(\overline{y_{i_k}}) \cdot G!R(\$)\, G!V(\overline{\$}) \\
u_{|R} &= R?G(b_1)\, R!V(b_1) \cdots R?G(b_m)\, R!V(b_m) \cdot \alpha \\
u_{|V} &= V?R(a_1)\, V?G(\overline{a_1}) \cdots V?R(a_n)\, V?G(\overline{a_n}) \cdot V?R(\$)\, V?G(\overline{\$})
\end{aligned}$$

for some $i_1, \ldots, i_k \in \{1, \ldots, K\}$ with $k \geq 1$ and $i_1 = 1$, some $a_1, \ldots, a_n \in \Sigma$ with $n \geq 0$, some $b_1, \ldots, b_m \in (\Sigma \cup \{\$, \#\})$ with $m \geq 0$, and some α that is either ε or a receive action $R?G(\cdot)$. The word $a_1\overline{a_1} \cdots a_n\overline{a_n}\$\overline{\$}$ received by V is a prefix of a shuffle of the word $b_1 \cdots b_m$ sent by R to V and of the word $\overline{y_{i_1}} \cdots \overline{y_{i_k}}\overline{\$}$ sent by G to V. So $a_1 \cdots a_n\$$ is a prefix of $b_1 \cdots b_m$ and $\overline{a_1} \cdots \overline{a_n}\overline{\$}$ is a prefix of $\overline{y_{i_1}} \cdots \overline{y_{i_k}}\overline{\$}$. Since $\overline{\$}$ does not occur in $\overline{y_{i_1}} \cdots \overline{y_{i_k}}$, we get that $a_1 \cdots a_n = y_{i_1} \cdots y_{i_k}$. Similarly, the word $b_1 \cdots b_m$ received by R is a prefix of the word $x_{i_1} \cdots x_{i_k}\$$ sent by G to R. We derive that $a_1 \cdots a_n\$$ is a prefix of $x_{i_1} \cdots x_{i_k}\$$. Since $\$$ does not occur in $x_{i_1} \cdots x_{i_k}$, this entails that $a_1 \cdots a_n\$ = b_1 \cdots b_m = x_{i_1} \cdots x_{i_k}\$$. We have shown that $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$. It remains to prove that $|x_{i_1} \cdots x_{i_j}| \geq |y_{i_1} \cdots y_{i_j}|$ for all $j \in \{1, \ldots, k\}$. Before that, we observe that every send action to V is matched in u. Indeed, the length of the word $a_1\overline{a_1} \cdots a_n\overline{a_n}\$\overline{\$}$ received by V is $2(n+1)$, and the length of the word sent to V is $|b_1 \cdots b_m| + |\overline{y_{i_1}} \cdots \overline{y_{i_k}}\overline{\$}| = 2m = 2(n+1)$. This means that the sequence of send actions to V in u is $R!V(b_1)\, G!V(\overline{b_1}) \cdots R!V(b_m)\, G!V(\overline{b_m})$.

We now show that $|x_{i_1} \cdots x_{i_j}| \geq |y_{i_1} \cdots y_{i_j}|$ for all $j \in \{1, \ldots, k\}$. Let $j \in \{1, \ldots, k\}$ and consider a prefix u_j of u such that ${u_j}_{|G} = G!R(x_{i_1})\, G!V(\overline{y_{i_1}}) \cdots G!R(x_{i_j})\, G!V(\overline{y_{i_j}})$. The projection of u_j on Act_R may be written as ${u_j}_{|R} = R?G(b_1)\, R!V(b_1) \cdots R?G(b_h)\, R!V(b_h) \cdot \beta$ for some $h \in \{0, \ldots, m\}$ and some β that is either ε or a receive action $R?G(\cdot)$. The word $b_1 \cdots b_h$ received by R is a prefix of the word $x_{i_1} \cdots x_{i_j}$ sent by G to R, so $|x_{i_1} \cdots x_{i_j}| \geq h$. Now, the word sent to V in u_j, say z, is a shuffle of the word $b_1 \cdots b_h$ sent by R to V and of the word $\overline{y_{i_1}} \cdots \overline{y_{i_j}}$ sent by G to V. The word z is also a prefix of the word $b_1\overline{b_1} \cdots b_m\overline{b_m}$ sent to V in u. This entails that $|b_1 \cdots b_h| = |\overline{y_{i_1}} \cdots \overline{y_{i_j}}|$ or $|b_1 \cdots b_h| = |\overline{y_{i_1}} \cdots \overline{y_{i_j}}| + 1$. It follows that $h \geq |y_{i_1} \cdots y_{i_j}|$ and we conclude that $|x_{i_1} \cdots x_{i_j}| \geq h \geq |y_{i_1} \cdots y_{i_j}|$. ◀

▶ **Lemma 3.4.** *For every trace $u \in Tr(\mathcal{A}(\mathcal{I}))$, if u does not contain the action $R!V(\#)$ then $match(u|_S)_{|p} \in L(\mathcal{A}_p)$ for every process $p \in \{G, R, V\}$.*

Proof. Let $u \in Tr(\mathcal{A}(\mathcal{I}))$ such that u does not contain the action $R!V(\#)$. We consider each process in $\mathbb{P} = \{G, R, V\}$ separately. Recall that the LTSes $\mathcal{A}_G$, $\mathcal{A}_R$ and $\mathcal{A}_V$ of the processes G, R and V are defined in Figures 2–4.

Let us start with the guesser process. According to the definition of $\mathcal{A}_G$, there exists a sequence of indices $i_1, \ldots, i_k \in \{1, \ldots, K\}$, with $k \geq 1$ and $i_1 = 1$, such that the sequence of send actions performed by G in u is a prefix of $v \cdot G!R(\$)\, G!V(\overline{\$})\, G!R(\#)$, where $v = G!R(x_{i_1})\, G!V(\overline{y_{i_1}}) \cdots G!R(x_{i_k})\, G!V(\overline{y_{i_k}})$. According to the definition of $\mathcal{A}_R$ and $\mathcal{A}_V$, the sequence of send actions to G in u is either empty or the single action $V!G(\#)$. Moreover, if

u contains $V!G(\#)$ then this action is necessarily after $G!V(\overline{\$})$ and before $G!R(\#)$. This is due to the receive actions $V?G(\overline{\$})$ and $G?V(\#)$. We derive that $match(u|_S)_{|G}$ is a prefix of $v \cdot G!R(\$)\, G!V(\overline{\$})\, G?V(\#)\, G!R(\#)$ and conclude that $match(u|_S)_{|G} \in L(\mathcal{A}_G)$.

We now proceed to the relayer process. According to the definition of $\mathcal{A}_G$ and $\mathcal{A}_V$, the sequence of send actions to R in u is a prefix of a sequence of the form $G!R(a_1) \cdots G!R(a_n) \cdot G!R(\$)\, G!R(\#)$, with $n \geq 0$ and $a_i \in \Sigma$. So the word sent to R in u is a prefix of $a_1 \cdots a_n \$\#$. This entails that the sequence of send actions performed by R in u is in the language $\{R!V(a) \mid a \in \Sigma\}^* \cdot \{\varepsilon, R!V(\$)\}$, according to the definition of $\mathcal{A}_R$. Indeed, either R selects the normal mode, in which case the word sent by R to V is a prefix of the word $a_1 \cdots a_n \$$ (recall that u does not contain the action $R!V(\#)$ by assumption), or R selects the dummy mode, in which case the word sent by R to V is in the language $\Sigma^* \cdot \{\varepsilon, \$\}$. We derive that the projection of u on sends by R, or to R, is in the shuffle of the language $\{R!V(a) \mid a \in \Sigma\}^* \cdot \{\varepsilon, R!V(\$)\}$ and the language $\{G!R(m) \mid m \in \Sigma \cup \{\$, \#\}\}^*$. It follows that $match(u|_S)_{|R}$ is in the shuffle of the language $\{R!V(a) \mid a \in \Sigma\}^* \cdot \{\varepsilon, R!V(\$)\}$ and the language $\{R?G(m) \mid m \in \Sigma \cup \{\$, \#\}\}^*$. We conclude that $match(u|_S)_{|R} \in L(\mathcal{A}_R)$, by letting R select the dummy mode.

Let us finally address the verifier process. We consider two cases, depending on whether u contains the action $V!G(\#)$ or not. If u does not contain the action $V!G(\#)$, then V does not perform any send action in u, according to the definition of $\mathcal{A}_V$. The projection of u on sends to V is in the language $(\{G!V(\overline{m}) \mid m \in \Sigma \cup \{\$\}\} \cup \{R!V(m) \mid m \in \Sigma \cup \{\$\}\})^*$, according to the definition of $\mathcal{A}_G$ and $\mathcal{A}_R$ (recall that u does not contain the action $R!V(\#)$ by assumption). It follows that $match(u|_S)_{|V}$ is in the language $(\{V?G(\overline{m}) \mid m \in \Sigma \cup \{\$\}\} \cup \{V?R(m) \mid m \in \Sigma \cup \{\$\}\})^*$. We conclude that $match(u|_S)_{|V} \in L(\mathcal{A}_V)$, by letting V select the dummy mode.

Assume now that u contains the action $V!G(\#)$. According to the definition of $\mathcal{A}_V$, this means that $u_{|V}$ is equal to $V?R(a_1)\, V?G(\overline{a_1}) \cdots V?R(a_n)\, V?G(\overline{a_n}) \cdot V?R(\$)\, V?G(\overline{\$})\, V!G(\#)$ for some $a_1, \ldots, a_n \in \Sigma$ with $n \geq 0$. The mailbox semantics entails that the word sent to V in u admits $a_1\overline{a_1} \cdots a_n\overline{a_n}\$\overline{\$}$ as a prefix. When we dealt about the relayer process, we showed that the word sent by R to V is in the language $\Sigma^* \cdot \{\varepsilon, \$\}$. Moreover, according to the definition of $\mathcal{A}_G$, the word sent by G to V is in the language $\overline{\Sigma}^* \cdot \{\varepsilon, \overline{\$}\}$. So the word sent to V in u is exactly $a_1\overline{a_1} \cdots a_n\overline{a_n}\$\overline{\$}$. We derive that the sequence of send actions to V in u is $v = R!V(a_1)\, G!V(\overline{a_1}) \cdots R!V(a_n)\, G!V(\overline{a_n}) \cdot R!V(\$)\, G!V(\overline{\$})$. Moreover, the single send action performed by V, namely $V!G(\#)$, is necessarily after $G!V(\overline{\$})$. This is due to the receive action $V?G(\overline{\$})$. It follows that $match(u|_S)_{|V}$ is equal to $V?R(a_1)\, V?G(\overline{a_1}) \cdots V?R(a_n)\, V?G(\overline{a_n}) \cdot V?R(\$)\, V?G(\overline{\$})\, V!G(\#)$, hence, $match(u|_S)_{|V} = u_{|V}$. We conclude that $match(u|_S)_{|V} \in L(\mathcal{A}_V)$. ◀

▶ **Corollary 3.5.** *If $\mathcal{I}$ is a negative instance of* Pre-MPCP *then $\mathcal{A}(\mathcal{I})$ is send-synchronizable.*

Proof. If $\mathcal{I}$ is a negative instance of Pre-MPCP then we derive from Lemma 3.3 that no trace in $Tr(\mathcal{A}(\mathcal{I}))$ contains the action $R!V(\#)$. It follows from Lemma 3.4 that $match(u|_S)_{|p} \in L(\mathcal{A}_p)$ for every trace $u \in Tr(\mathcal{A}(\mathcal{I}))$ and every process $p \in \{G, R, V\}$. We conclude thanks to Lemma B.1 that $\mathcal{A}(\mathcal{I})$ is send-synchronizable. ◀