

Demystifying asynchronous communication and its variants*

Subtitle†

ANONYMOUS AUTHOR(S)

Text of abstract

Additional Key Words and Phrases: keyword1, keyword2, keyword3

ACM Reference Format:

Anonymous Author(s). 2018. Demystifying asynchronous communication and its variants: Subtitle. *Proc. ACM Program. Lang.* 1, CONF, Article 1 (January 2018), 19 pages.

1 INTRODUCTION

- Interleaving based semantics VS partial order/graph based semantics
- Synchronous and asynchronous communication
- The problem of synchronizability

2 PRELIMINARIES/BASICS

- Communicating systems (communicating finite-state automata with bag channels)
- MSCs and conflict graph
- Monadic Second-Order logic on MSCs
- (Language of a system as a set of MSCs)
- (Model checking and synchronizability)

3 ASYNCHRONOUS COMMUNICATION MODELS OVERVIEW

- Overview of asynchronous variants
- High-level description of each variant along with references to implementations (if existing)
- (Definitions based on linearization, intuitive)
- (Language of a system with a given communication model as a set of MSCs)
- Hint of hierarchy result

4 ASYNCHRONOUS COMMUNICATION MODELS OPERATIONAL SEMANTICS

- TODO...

5 ASYNCHRONOUS COMMUNICATION MODELS AS CLASSES OF MSCS, MSO-DEFINABILITY

- Definition of MSC class for each communication model (alternative definitions)
- MSO-definability of each class

6 EQUIVALENCE OF THE TWO DEFINITIONS

- TODO...

7 HIERARCHY OF ASYNCHRONOUS CLASSES OF MSCS

...

*Title note

†Subtitle note

2018. 2475-1421/2018/1-ART1 \$15.00

<https://doi.org/>

8 AN APPLICATION: SPECIAL TREEWIDTH AND DECIDABILITY OF THE SYNCHRONIZABILITY PROBLEM

- The synchronizability problem
- Special treewidth and how the results regarding the hierarchy are useful for detecting STW-boundness of certain classes
- MSO-decidability and STW-boundness tables

9 CONCLUSION

A APPENDIX

Text of appendix ...

B (3) ASYNCHRONOUS COMMUNICATION MODELS OVERVIEW

In synchronous communication, send and receive elements are essentially viewed as a single entity, i.e. a receive event is always executed immediately after the corresponding send event. The whole idea behind (fully) asynchronous communication is to decouple send and receive events, so that a receive event can happen indefinitely after its corresponding send event. However, by introducing some additional constraints on asynchronous communication, we can obtain new communication models that sit somewhere between synchronous and fully asynchronous communication. In this section, we will present 7 different asynchronous communication models, which were already introduced in [Chevrou et al. 2016] (even though they do not consider unmatched messages). A major difference is that in [Chevrou et al. 2016] these communication models are addressed from a linearizations standpoint, whereas we are interested in MSCs. Recall that a single MSC can have several possible linearizations. The work in [Chevrou et al. 2016] describes the properties that a single linearization must satisfy in order to be realizable by a system that uses a given communication model. On the other hand, we are interested in understanding if a given MSC describes a computation that can be realized by a system that uses some communication model CM . In other words, given a MSC we want to know if it has at least one linearization that respects the constraints imposed by CM . If that is the case, the MSC represents a behaviour that can be exhibited by a system that uses CM as a communication model. These are two fundamentally dissimilar problems; at the end of this section we provide an example to clarify the difference. In our work, we are going to formally characterize the classes of MSCs which represent valid computations for all of these 7 asynchronous communication model. We also show how these classes form a well-defined hierarchy, which does not correspond entirely to that found in [Chevrou et al. 2016].

We model a distributed system as a set of concurrent Finite-State Machines (FSMs) that exchange messages asynchronously through channels. Each FSM models a single machine/process of the system and transitions are labeled with "send" and "receive" operations, which specify the sender and the receiver of a message. In our work we consider only point-to-point communication, that is, messages that have exactly one sender and one receiver. The role of the communication model is to impose an order on the reception of messages, according to its specification. For instance, the delivery of a message could be delayed or even prevented by a communication model CM , so as to ensure that messages are received in an order that is valid for CM . The 7 communication models that we address all impose different constraints on the order in which messages can be received.

B.1 Fully asynchronous

In the fully asynchronous communication model (or simply asynchronous) messages can be received at any time once they have been sent. In asynchronous communication, send events are non-blocking, i.e. the sender of a message does not have to wait for it to be delivered to the recipient, in order to resume normal operations. Fig. 1 shows a computation that can be executed by a system that uses asynchronous communication; indeed, even if m_1 is sent before m_2 , q does not have to receive m_1 first. For convenience, we will refer to a system that uses asynchronous communication simply as an asynchronous system. In a similar way, an MSC such that in Fig. 1 will be referred to as an asynchronous MSC, since it represents a valid computation for an asynchronous system. The same jargon will also be used for all the other communication models. We will call MSC_{asy} the set of all asynchronous MSCs.

B.2 FIFO 1–1 (p2p)

In the FIFO 1–1 communication model, any two messages sent from one process p to another process q are always received in the same order as they were sent. In the most classical definition

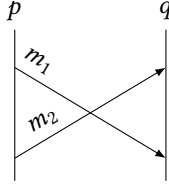


Fig. 1. An asynchronous MSC.

of Communicating Finite-State Machine, processes are connected pairwise by FIFO channels, i.e. messages are delivered by channels in the order in which they were sent¹. This definition of Communicating Finite-State Machines clearly uses the FIFO 1–1 communication model, since we have FIFO channels between processes that take care of delivering messages in the correct order. The FIFO 1–1 communication model is referred to as p2p in [Bollig et al. 2021], and we will also use this terminology. The MSC shown in Fig. 1 is clearly not a FIFO 1–1 MSC; both m_1 and m_2 are sent by and to the same process, so the receive order must match the send order, which is not the case here. Fig. 2 shows an example of p2p MSC; the only two messages sent by and to the same process are m_3 and m_4 , which are received in the same order as they have been sent. An example of linearization is !1 !2 ?2 !3 !4 ?3 ?1 ?4. Let MSC_{p2p} be the set of p2p MSCs.

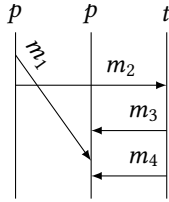


Fig. 2. A p2p MSC.

B.3 Causally ordered

In the causally ordered communication model, messages are delivered to a process according to the causality of their emissions. In other words, if there are two messages m_1 and m_2 with the same recipient, such that m_1 is causally sent before m_2 (i.e. there exists a causal path from the first send to the second one), then m_1 must be received before m_2 . Fig. 3a, which is identical to Fig. 2, shows an example of non-causally ordered MSC; there is a causal path between the sending of m_1 and m_3 (highlighted with red arrows), hence m_1 should be received before m_3 according to the causally ordered communication model, which is not the case. On the other hand, the second MSC shown in Fig. 3 is causally ordered; note that the only two messages with the same recipient are m_2 and m_3 , but there is no causal path between their respective send events (i.e. the causally ordered communication model does not introduce any new constraint that must be satisfied). Let MSC_{co} be the set of causally ordered MSCs.

¹Please note that our definition of Communicating Finite-State Machine is different from the classical one. FIFO channels are replaced by bag channels.



(a) Asynchronous, p2p, not causally ordered, not mailbox, not FIFO 1-n, not FIFO n-n, not RSC. (b) Asynchronous, p2p, causally ordered, mailbox, FIFO 1-n, FIFO n-n, not RSC.

Fig. 3. Two examples of MSCs.

B.4 FIFO $n-1$ (mailbox)

In the FIFO $n-1$ communicating model, any two messages sent to a process q must be received in the same order as they have been sent (according to absolute time). Note that these two messages might be sent by different processes and the two send events might be concurrent (i.e. there is no causal path between them). In other words, if a process q receives m_1 before m_2 , then m_1 must have been sent before m_2 in absolute time. Essentially, the FIFO $n-1$ coordinates all the senders of a single receiver. A high-level implementation of the mailbox communication model could consist in a single incoming FIFO channel for each process, which is shared by all the other processes. A send event would consist in pushing the message on the shared FIFO channel. The MSC shown in Fig. 3a is not a mailbox MSC; m_1 and m_3 have the same recipient, but they are not received in the same order as they are sent. The MSC in Fig. 3b is mailbox; indeed, we are able to find a linearization that respects the mailbox constraints, such as !1 !2 !3 ?2 ?3 ?1 (note that m_2 is both sent and received before m_3). Such a linearization will be referred to as a *mailbox linearization*. At this stage, the difference between the class of causally ordered MSCs and the class of mailbox MSCs might not be clear. We will clarify later how all these classes of MSCs are related to each other. Let MSC_{mb} be the set of mailbox MSCs.

B.5 FIFO $1-n$

The FIFO $1-n$ communicating model is the dual of FIFO $n-1$, it coordinates a sender with all the receivers. Any two messages sent by a process p must be received in the same order (in absolute time) as they have been sent. Note that these two messages might be received by different processes and the two receive events might be concurrent (i.e. there is no causal path between them). In other words, if a process p sends m_1 before m_2 , then m_1 must be received before m_2 in absolute time. A high-level implementation of the FIFO $1-n$ communication model could consist in a single outgoing FIFO channel for each process P_i , which is shared by all the other processes. A send event would consist in pushing the message on the outgoing FIFO channel. The MSC shown in Fig. 3a is not a FIFO $1-n$ MSC; m_1 and m_2 are sent in this order by the same process, but they are received in the opposite order (note that there is a causal path between the reception of m_2 and the reception of m_1 , so ?2 happens before ?1 in every linearization of this MSC). Fig. 3b shows an example of FIFO $1-n$ MSC; m_1 is sent before m_2 by the same process, and we are able to find a linearization where m_1 is received before m_2 , such as !1 !2 !3 ?1 ?2 ?3. Such a linearization will be referred to as a *1-n linearization*. Let MSC_{1-n} be the set of $1-n$ MSCs.

B.6 FIFO n - n

In the FIFO n - n communicating model, messages are globally ordered and delivered according to their emission order. Any two messages must be received in the same order as they have been sent, in absolute time. Note that these two messages might be received by different processes and the two receive events might be concurrent (i.e. there is no causal path between them). In other words, if a message m_1 is sent before m_2 in absolute time, then m_1 must be received before m_2 in absolute time. The FIFO n - n coordinates all the senders with all the receivers. A high-level implementation of the FIFO 1 - n communication model could consist in a single FIFO channel shared by all processes. The MSC shown in Fig. 3a is clearly not a FIFO n - n MSC; if we consider messages m_1 and m_2 we have that, in every linearization, !1 happens before !2 and ?2 happens before ?1. This violates the constraints imposed by the FIFO n - n communication model. The MSC in Fig. 3b is n - n because we are able to find a linearization that satisfies the n - n communication model, e.g. !1 !2 !3 ?1 ?2 ?3. Such a linearization will be referred to as an n - n linearization. Let MSC_{n-n} be the set of n - n MSCs.

B.7 Realizable with Synchronous Communication (RSC)

The RSC communication model imposes that a send event is always immediately followed by its corresponding receive event. In an execution of a system that uses the RSC communication model it is impossible to find an event that is executed between a send and its corresponding receive. An asynchronous distributed system that implements the RSC communication model effectively behaves as a synchronous system. None of the MSCs shown in Fig. 3 is a RSC MSC; indeed, for both of them it is impossible to find a linearization where each send event is immediately followed by the corresponding receive event. The MSC shown in Fig. 4 is an example of RSC MSC; we can easily find a linearization that respects the constraints of the RSC communication model, such as !1 ?1 !2 ?2 !3 ?3. Such a linearization will be referred to as an RSC linearization. Let MSC_{RSC} be the set of RSC MSCs.

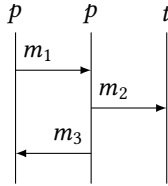


Fig. 4. A RSC MSC.

B.8 Hierarchy of MSC classes

We find of particular interest to study the relation between the classes of MSCs for all of these communication models. For instance, it is trivial to see that each p2p MSC is also asynchronous (i.e. $MSC_{p2p} \subseteq MSC_{asy}$); p2p communication is equivalent to asynchronous communication plus the FIFO constraints, therefore every MSC that respects the p2p communication model is also valid for the asynchronous communication model. It is also quite straightforward to notice that every causally ordered MSC is a p2p MSC. In Section ?? we prove that all these classes of MSCs form a very neat hierarchy, which is graphically shown in Fig. 5.

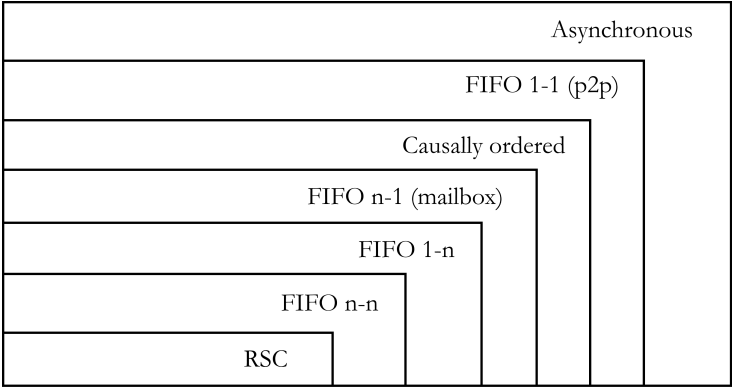


Fig. 5. The hierarchy of MSC classes.

C (5) ASYNCHRONOUS COMMUNICATION MODELS AS CLASSES OF MSCS, MSO-DEFINABILITY

C.1 Definitions

A Message Sequence Chart (MSC), such as the one in Fig. ??, provides a visual description of the behaviour of a distributed system. In this section, we start by formally defining the most generic class of Message Sequence Charts (MSCs), which we call asynchronous MSCs. More specialized classes of MSCs, such as $p2p$ MSCs, will also be discussed. Intuitively, we say that an MSC M is asynchronous if there is an asynchronous system \mathcal{S} that can exhibit the behaviour described by M .

Definition C.1 (Asynchronous MSC). An *asynchronous MSC* (or simply MSC) over \mathbb{P} and \mathbb{M} is a tuple $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$, where \mathcal{E} is a finite (possibly empty) set of *events* and $\lambda : \mathcal{E} \rightarrow \Sigma$ is a labeling function that associates an action to each event. For $p \in \mathbb{P}$, let $\mathcal{E}_p = \{e \in \mathcal{E} \mid \lambda(e) \in \Sigma_p\}$ be the set of events that are executed by p . We require that \rightarrow (the *process relation*) is the disjoint union $\bigcup_{p \in \mathbb{P}} \rightarrow_p$ of relations $\rightarrow_p \subseteq \mathcal{E}_p \times \mathcal{E}_p$ such that \rightarrow_p is the direct successor relation of a total order on \mathcal{E}_p . For an event $e \in \mathcal{E}$, a set of actions $A \subseteq \Sigma$, and a relation $R \subseteq \mathcal{E} \times \mathcal{E}$, let $\#_A(R, e) = |\{f \in \mathcal{E} \mid (f, e) \in R \text{ and } \lambda(f) \in A\}|$. We require that $\triangleleft \subseteq \mathcal{E} \times \mathcal{E}$ (the *message relation*) satisfies the following:

- (1) for every pair $(e, f) \in \triangleleft$, there is a send action $send(p, q, m) \in \Sigma$ such that $\lambda(e) = send(p, q, m)$, $\lambda(f) = rec(p, q, m)$.
- (2) for all $f \in \mathcal{E}$ such that $\lambda(f)$ is a receive action, there is exactly one $e \in \mathcal{E}$ such that $e \triangleleft f$.

Finally, letting $\leq_M = (\rightarrow \cup \triangleleft)^*$, we require that \leq_M is a partial order. For convenience, we simply write \leq when M is clear from the context. We will refer to \leq as the *causal ordering* or *happens-before* relation. If, for two events e and f , we have that $e \leq f$, we will equivalently say that there is a *causal path* between e and f .

According to Condition (2), every receive event must have a matching send event. Note that, however, there may be unmatched send events. We let $SendEv(M) = \{e \in \mathcal{E} \mid \lambda(e) \text{ is a send action}\}$, $RecEv(M) = \{e \in \mathcal{E} \mid \lambda(e) \text{ is a receive action}\}$, $Matched(M) = \{e \in \mathcal{E} \mid \text{there is } f \in \mathcal{E} \text{ such that } e \triangleleft f\}$, and $Unm(M) = \{e \in \mathcal{E} \mid \lambda(e) \text{ is a send action and there is no } f \in \mathcal{E} \text{ such that } e \triangleleft f\}$. We do not distinguish isomorphic MSCs and let MSC_{asy} be the set of all the asynchronous MSCs over the given sets \mathbb{P} and \mathbb{M} .

Linearizations. Consider $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda) \in MSC_{asy}$. A *linearization* of M is a (reflexive) total order $\rightsquigarrow \subseteq \mathcal{E} \times \mathcal{E}$ such that $\leq_M \subseteq \rightsquigarrow$. In other words, a linearization of M is a total order that respects the happens-before relation \leq_M defined over M .

D: Provide example of linearization.

Asynchronous MSCs are the widest class of MSCs that we will deal with. By introducing additional constraints, we are able to define other classes of MSCs that exclusively describe the behaviours of $p2p$ systems, mailbox systems, and so on.

D: Provide example of asynchronous MSC that is not $p2p$.

As in the asynchronous case, we say that M is a $p2p$ MSC if there is a $p2p$ system that can produce the behaviour described by M . We give here the formal definition of $p2p$ MSC, which also considers the possibility of having unmatched messages (i.e. messages that are sent but not received).

D: Why unmatched messages? What do they represent? When an automata does not have the receive action for a message that was already sent (see examples at the end of concur paper).

Definition C.2 (Peer-to-peer MSCs). A *p2p MSC* (or simply *MSC*) is an asynchronous MSC where we require that, for every pair $(e, f) \in \triangleleft$, such that $\lambda(e) = \text{send}(p, q, m)$, $\lambda(f) = \text{rec}(p, q, m)$, we have $\#_{\text{Send}(p, q, _)}(\rightarrow^+, e) = \#_{\text{Rec}(p, q, _)}(\rightarrow^+, f)$.

The additional constraint satisfied by p2p MSCs ensures that channels operate in FIFO mode; when a process q receives a message from a process p , it must have already received all the messages that were previously sent to him by p . Let MSC_{p2p} denote the set of all the p2p MSCs over two given sets \mathbb{P} and \mathbb{M} . Note that, by definition, every p2p MSC is an asynchronous MSC. The idea is that we are always able to find an asynchronous system that *can* exhibit the behaviour described by a p2p MSC; after all, the channels of an asynchronous system do not have to follow any specific behaviour, so they can indeed happen to operate as if they were queues. Example ?? shows that the opposite direction is generally not true, an asynchronous MSC is not always a p2p MSC. It follows that $\text{MSC}_{\text{p2p}} \subset \text{MSC}_{\text{asy}}$.

We will now consider the class of MSCs for which there is a causally ordered system that can produce their behaviour. Intuitively, an MSC is causally ordered if all the messages sent to the same process are received in an order which is consistent with the causal ordering of the corresponding send events. Below the formal definition, which also considers unmatched messages.

D: Provide example of an MSC that is not causally ordered

Definition C.3 (Causally ordered MSC). An MSC $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$ is *causally ordered* if, for any two send events s and s' , such that $\lambda(s) = \text{Send}(_, q, _)$, $\lambda(s') = \text{Send}(_, q, _)$, and $s \leq_M s'$, we have either:

- $s, s' \in \text{Matched}(M)$ and $r \rightarrow^* r'$, where r and r' are two receive events such that $s \triangleleft r$ and $s' \triangleleft r'$.
- $s' \in \text{Unm}(M)$.

By definition, every causally ordered MSC is a p2p MSC. This is not surprising, considering that a causally ordered system is essentially a p2p system with an additional constraint on the delivery of messages; indeed, we are always able to find a p2p system that *can* exhibit the behaviour described by a causally ordered MSC. Let MSC_{co} denote the set of all the causally ordered MSCs over two given sets \mathbb{P} and \mathbb{M} . Example ?? shows that a p2p MSC is not always a causally ordered MSC. It follows that $\text{MSC}_{\text{co}} \subset \text{MSC}_{\text{p2p}}$.

Moving on to the mailbox semantics, we say that M is a mailbox MSC if there is a mailbox system that can exhibit the behaviour described by M .

D: Provide example of MSC which is not mailbox

Definition C.4 (Mailbox MSC). An MSC $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$ is a *mailbox MSC* if it has a linearization \rightsquigarrow where, for any two send events s and s' , such that $\lambda(s) = \text{Send}(_, q, _)$, $\lambda(s') = \text{Send}(_, q, _)$, and $s \rightsquigarrow s'$, we have either:

- $s, s' \in \text{Matched}(M)$ and $r \rightsquigarrow r'$, where r and r' are two receive events such that $s \triangleleft r$ and $s' \triangleleft r'$.
- $s' \in \text{Unm}(M)$.

Such a linearization will be referred to as a *mailbox linearization*, and the symbol $\rightsquigarrow^{\text{mb}}$ will be used to denote one. Let MSC_{mb} denote the set of all the mailbox MSCs over two given sets \mathbb{P} and \mathbb{M} . By definition, every mailbox MSC is a p2p MSC. Conversely, Example ?? shows a p2p MSC which is not a mailbox MSC. It follows that $\text{MSC}_{\text{mb}} \subset \text{MSC}_{\text{p2p}}$. We show here that each mailbox MSC is also a causally ordered MSC.

D: Provide example of causally ordered MSC which is not mailbox (Figure 2.18 of Laetitia's thesis).

PROPOSITION C.5. *Every mailbox MSC is a causally ordered MSC.*

PROOF. Let M be a mailbox MSC and \rightsquigarrow a mailbox linearization of it. Recall that a linearization has to respect the happens-before partial order over M , i.e. $\leq_M \subseteq \rightsquigarrow$. Consider any two send events s and s' , such that $\lambda(s) = \text{Send}(_, q, _)$, $\lambda(s') = \text{Send}(_, q, _)$ and $s \leq_M s'$. Since $\leq_M \subseteq \rightsquigarrow$, we have that $s \rightsquigarrow s'$ and, by the definition of mailbox linearization, either (i) $s' \in \text{Unm}(M)$, or (ii) $s, s' \in \text{Matched}(M)$, $s \triangleleft r$, $s' \triangleleft r'$ and $r \rightsquigarrow r'$. The former clearly respects the definition of causally ordered MSC, so let us focus on the latter. Note that r and r' are two receive events executed by the same process, hence $r \rightsquigarrow r'$ implies $r \rightarrow^+ r'$. It follows that M is a causally ordered MSC. \square

Moving on to the 1- n semantics, we say that M is a 1- n MSC if there is a 1- n system that can exhibit the behaviour described by M .

Definition C.6 (1- n MSC). An MSC $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$ is a 1- n MSC if it has a linearization \rightsquigarrow where, for any two send events s and s' , such that $\lambda(s) = \text{Send}(p, _, _)$, $\lambda(s') = \text{Send}(p, _, _)$, and $s \rightarrow^+ s'$ (which implies $s \rightsquigarrow s'$), we have either:

- $s, s' \in \text{Matched}(M)$ and $r \rightsquigarrow r'$, where r and r' are two receive events such that $s \triangleleft r$ and $s' \triangleleft r'$.
- $s' \in \text{Unm}(M)$.

Such a linearization will be referred to as a 1- n *linearization*. Note that the definition is very similar to the mailbox case, but here s and s' are two send events executed by the same process. Let MSC_{1-n} denote the set of all the 1- n MSCs over two given sets \mathbb{P} and \mathbb{M} . By definition, every 1- n MSC is a p2p MSC. Conversely, Example ?? shows a p2p MSC which is not a 1- n MSC. It follows that $\text{MSC}_{\text{mb}} \subset \text{MSC}_{\text{p2p}}$. We show here that each 1- n MSC is also a causally ordered MSC, which is not as intuitive as the mailbox case.

D: Provide example of MSC which is not 1- n

D: Should I still leave this proof if we showed that every 1- n MSC is a mailbox MSC?

PROPOSITION C.7. *Every 1- n MSC is a causally ordered MSC.*

PROOF. By contradiction. Suppose that M is a 1- n MSC, but not a causally ordered MSC. Since M is not causally ordered, there must be two send events s and s' such that $\lambda(s) = \text{Send}(_, q, _)$, $\lambda(s') = \text{Send}(_, q, _)$, $s \leq_M s'$, and we have either:

- (1) $s, s' \in \text{Matched}(M)$ and $r' \rightarrow^* r$, where r and r' are two receive events such that $s \triangleleft r$ and $s' \triangleleft r'$.
- (2) $s \in \text{Unm}(M)$ and $s' \in \text{Matched}(M)$.

We need to show that both of these scenarios lead to a contradiction. (1) Suppose s and s' are executed by the same process. Since M is a 1- n MSC, there must be a linearization \rightsquigarrow such that $r \rightsquigarrow r'$, but this is clearly impossible since we have $r' \rightarrow^* r$. Suppose now that s and s' are executed

by two different processes p and q . We know by hypothesis that $s \leq_M s'$, i.e. there is a causal path of events $P = s \sim a \sim \dots \sim s' \sim r'$ from s to r' , where \sim is either \rightarrow or \triangleleft . Refer to the first example in Figure 6 for a visual representation (P is drawn in purple). To have a causal path P , there must be a send event s'' that is executed by p after s and that is part of P , along with its receipt r'' (i.e. $P = s \leq_M s'' \triangleleft r'' \leq_M s' \triangleleft r'$). We clearly have $r'' \rightsquigarrow r'$ for any linearization of M , because $r'' \leq_M r'$ (they are both in the causal path P and r'' happens before r). Since M is a 1- n MSC, there has to be a linearization \rightsquigarrow where $r \rightsquigarrow r''$, because s and s'' are send events executed by the same process. It follows that M should have a linearization where $r \rightsquigarrow r'' \rightsquigarrow r'$, but this is not possible because of the hypothesis that $r' \rightarrow^* r$. This is a contradiction. (2) Suppose s and s' are executed by the same process. It is trivial to see, by definition, that M cannot be a 1- n MSC. Suppose now that s and s' are executed by two different processes p and q , and consider the same send event s'' as before (executed by p). Refer to the second example in Figure 6 for a visual representation. Since s'' is matched, we have two events s and s'' , sent by the same process p , that are unmatched and matched, respectively. Clearly, M cannot be a 1- n MSC. \square

Fig. 6. Two examples of 1- n MSCs.

D: Provide example of causally ordered MSC which is not 1- n (same as mailbox! Figure 2.18 of Laetitia's thesis).

Definition C.8 (1- n alternative). For an MSC $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$, we define an additional binary relation that represents a constraint under the 1- n semantics, which ensures that messages sent from the same process are received in the same order. Let $\blacktriangleleft_M \subseteq \mathcal{E} \times \mathcal{E}$ be defined as $e_1 \blacktriangleleft_M e_2$ if there are two events e_1 and e_2 , and $p \in \mathbb{P}$ such that either:

- $\lambda(e_1) \in \text{Send}(p, _, _)$, $\lambda(e_2) \in \text{Send}(p, _, _)$, $e_1 \in \text{Matched}(M)$, and $e_2 \in \text{Unm}(M)$, or
- $\lambda(e_1) \in \text{Rec}(p, _, _)$, $\lambda(e_2) \in \text{Rec}(p, _, _)$, $s_1 \triangleleft e_1$ and $s_2 \triangleleft e_2$ for some $s_1, s_2 \in \mathcal{E}_p$, and $s_1 \rightarrow^+ s_2$.

We let $\leq_M = (\rightarrow \cup \triangleleft \cup \blacktriangleleft_M)^*$. Note that $\leq_M \subseteq \leq_M$. We call $M \in \text{MSC}_{\text{asy}}$ a 1- n MSC if \leq_M is a partial order.

D: Should I also include the proof that every mailbox MSC without unmatched messages is 1- n ? Do we need it?

PROPOSITION C.9. Every 1- n MSC without unmatched messages is a mailbox MSC.

PROOF. We show that the contrapositive is true, i.e. if an MSC is not mailbox (and it does not have unmatched messages), it is also not 1- n . Suppose M is an asynchronous MSC, but not mailbox. There must be a cycle ξ such that $e \leq e$, for some event e . Recall that $\leq = (\rightarrow \cup \triangleleft \cup \square)^*$ and $\leq = (\rightarrow \cup \triangleleft)^*$. We can always explicitly write a cycle $e \leq e$ only using \square and \leq . For instance, there might be a cycle $e \leq e$ because we have that $e \square f \leq g \square h \square i \leq e$. Consider any two

adjacent events s_1 and s_2 in the cycle ξ , where ξ has been written using only \sqsubset and \leq , and we never have two consecutive \leq ². We have two cases:

- (1) $s_1 \sqsubset s_2$. We know, by definition of \sqsubset , that s_1 and s_2 must be two send events and that $r_1 \rightarrow^+ r_2$, where r_1 and r_2 are the receive events that match with s_1 and s_2 , respectively (we are not considering unmatched messages by hypothesis).
- (2) $s_1 \leq s_2$. Since M is asynchronous by hypothesis, ξ has to contain at least one \sqsubset ³; recall that we also wrote ξ in such a way that we do not have two consecutive \leq . It is not difficult to see that s_1 and s_2 have to be send events, since they belong to ξ . We have two cases:
 - (a) r_1 is in the causal path, i.e. $s_1 \triangleleft r_1 \leq s_2$. In particular, note that $r_1 \leq r_2$.
 - (b) r_1 is not in the causal path, hence there must be a message m_k sent by the same process that sent s_1 , such that $s_1 \rightarrow^+ s_k \triangleleft r_k \leq s_2 \triangleleft r_2$, where s_k and r_k are the send and receive events associated with m_k , respectively. Since messages m_1 and m_k are sent by the same process and $s_1 \rightarrow^+ s_k$, we should have $r_1 \triangleleft r_k$, according to the 1- n semantics. In particular, note that we have $r_1 \triangleleft r_k \leq r_2$.

In both case (a) and (b), we conclude that $r_1 \triangleleft r_2$. Recall that $\triangleleft = (\rightarrow \cup \triangleleft \cup \triangleleft_M)^*$.

Notice that, for either cases, a relation between two send events s_1 and s_2 (i.e. $s_1 \sqsubset s_2$ or $s_1 \leq s_2$) always implies a relation between the respective receive events r_1 and r_2 , according to the 1- n semantics. It follows that ξ , which is a cycle for the \leq relation, always implies a cycle for the \triangleleft relation⁴, as shown by the following example. Let M be a non-mailbox MSC, and suppose we have a cycle $s_1 \sqsubset s_2 \sqsubset s_3 \leq s_4 \sqsubset s_5 \leq s_1$. $s_1 \sqsubset s_2$ falls into case (1), so it implies $r_1 \rightarrow^+ r_2$. The same goes for $s_2 \sqsubset r_3$, which implies $r_2 \rightarrow^+ r_3$. $s_3 \leq s_4$ falls into case (2), and implies that $r_3 \triangleleft r_4$. $s_4 \sqsubset s_5$ falls into case (1) and it implies $r_4 \rightarrow^+ r_5$. $s_5 \leq s_1$ falls into case (2) and implies that $r_5 \triangleleft r_1$. Putting all these implications together, we have that $r_1 \rightarrow^+ r_2 \rightarrow^+ r_3 \triangleleft r_4 \rightarrow^+ r_5 \triangleleft r_1$, which is a cycle for \triangleleft . Note that, given any cycle for \leq , we are always able to apply this technique to obtain a cycle for \triangleleft . \square

Proposition C.9 remains true even if we consider unmatched messages.

PROPOSITION C.10. *Every 1- n MSC is a mailbox MSC.*

PROOF. Let M be an asynchronous MSC. The proof proceeds in the same way as the one of Proposition C.9, but unmatched messages introduce some additional cases. Consider any two adjacent events s_1 and s_2 in a cycle ξ for \leq , where ξ has been written using only \sqsubset and \leq , and we never have two consecutive \leq . These are some additional cases:

- (3) $u_1 \sqsubset s_2$, where u_1 is the send event of an unmatched message. This case never happens because of how \sqsubset is defined.
- (4) $u_1 \leq u_2$, where u_1 and u_2 are both send events of unmatched messages. Since both u_1 and u_2 are part of the cycle ξ , there must be an event s_3 such that $u_1 \leq u_2 \sqsubset s_3$. However, $u_2 \sqsubset s_3$ falls into case (3), which can never happen.
- (5) $u_1 \leq s_2$, where u_1 is the send event of an unmatched message and s_2 is the send event of a matched message. Since we have a causal path between u_1 and s_2 , there has to be a message m_k , sent by the same process that sent m_1 , such that $u_1 \rightarrow^+ s_k \triangleleft r_k \leq s_2 \triangleleft r_2$ ⁵, where s_k and r_k are the send and receive events associated with m_k , respectively. Since messages m_1 and m_k are sent by the same process and m_1 is unmatched, we should have $s_k \triangleleft u_1$, according to

²This is always possible, since $a \leq b \leq c$ is written as $a \leq c$.

³If that was not the case, \leq would also be cyclic and M would not be an asynchronous MSC.

⁴If \triangleleft is cyclic, M is not a 1- n MSC.

⁵Note that we can have $m_k = m_2$

the 1- n semantics, but $u_1 \rightarrow^+ s_k$. It follows that if ξ contains $u_1 \leq s_2$, we can immediately conclude that M is not a 1- n MSC.

- (6) $s_1 \sqsubset u_2$, where s_1 is the send event of a matched message and u_2 is the send event of an unmatched message. Since both s_1 and u_2 are part of a cycle, there must be an event s_3 such that $s_1 \sqsubset u_2 \leq s_3$; we cannot have $u_2 \sqsubset s_3$, because of case (3). $u_2 \leq s_3$ falls into case (5), so we can conclude that M is not a 1- n MSC.

We showed that cases (3) and (4) can never happen, whereas cases (5) and (6) both imply that M is not 1- n . If we combine them with the cases described in Proposition C.9 we have the full proof. \square

Definition C.11 (n - n MSC). An MSC $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$ is a n - n MSC if it has a linearization \rightsquigarrow where, for any two send events s and s' , such that $s \rightsquigarrow s'$, we have either:

- $s, s' \in \text{Matched}(M)$ and $r \rightsquigarrow r'$, where r and r' are two receive events such that $s \triangleleft r$ and $s' \triangleleft r'$.
- $s' \in \text{Unm}(M)$.

Such a linearization will be referred to as a n - n linearization. Intuitively, with an n - n MSC we are always able to schedule events in such a way that messages are received in the same order as they were sent, and unmatched messages are sent only after all matched messages are sent. By definition, every n - n MSC is a 1- n MSC.

Definition C.12 (n - n alternative). For an MSC $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$, let $\preceq_M = (\rightarrow \cup \triangleleft \cup \sqsubset_M \cup \blacktriangleleft_M)^*$. We define an additional binary relation $\bowtie_M \subseteq \mathcal{E} \times \mathcal{E}$, such that for two events e_1 and e_2 we have $e_1 \bowtie_M e_2$ if one of the following holds:

- (1) $e_1 \preceq_M e_2$
- (2) $\lambda(e_1) \in \text{Rec}(_, _, _)$, $\lambda(e_2) \in \text{Rec}(_, _, _)$, $s_1 \triangleleft e_1$ and $s_2 \triangleleft e_2$ for some $s_1, s_2 \in \mathcal{E}$, $s_1 \preceq_M s_2$ and $e_1 \not\bowtie_M e_2$.
- (3) $\lambda(e_1) \in \text{Send}(_, _, _)$, $\lambda(e_2) \in \text{Send}(_, _, _)$, $e_1 \triangleleft r_1$ and $e_2 \triangleleft r_2$ for some $r_1, r_2 \in \mathcal{E}$, $r_1 \preceq_M r_2$ and $e_1 \not\bowtie_M e_2$.
- (4) $e_1 \in \text{Matched}(M)$, $e_2 \in \text{Unm}(M)$, $e_1 \not\bowtie_M e_2$.

Note that $\leq_M \subseteq \preceq_M$, $\trianglelefteq_M \subseteq \preceq_M$, and $\preceq_M \subseteq \bowtie_M$. We call $M \in \text{MSC}_{\text{asy}}$ a n - n MSC if \bowtie_M is acyclic.

D: Initially I said "We call $M \in \text{MSC}_{\text{asy}}$ a n - n MSC if \bowtie_M is a partial order", but I don't think it is true, since \bowtie_M does not have to be transitive.

It is not trivial to see that Definition C.12 is equivalent to Definition C.11. To show that, we need some preliminary results and definitions.

PROPOSITION C.13. *Let M be an MSC. Given two matched send events s_1 and s_2 , and their respective receive events r_1 and r_2 , $r_1 \bowtie_M r_2 \implies s_1 \bowtie_M s_2$.*

PROOF. Follows from the definition of \bowtie_M . We have $r_1 \bowtie_M r_2$ if either:

- $r_1 \preceq_M r_2$. Two cases: either (i) $s_1 \preceq_M s_2$, or (ii) $s_1 \not\bowtie_M s_2$. The first case clearly implies $s_1 \bowtie_M s_2$, for rule 1 in the definition of \bowtie_M . The second too, because of rule 3.
- $r_1 \not\bowtie_M r_2$, but $r_1 \preceq_M r_2$. This is only possible if rule 2 in the definition of \bowtie_M was used, which implies $s_1 \preceq_M s_2$ and, for rule 1, $s_1 \bowtie_M s_2$.

\square

PROPOSITION C.14. *Let M be an MSC. If \bowtie_M is cyclic, then M is not n - n .*

PROOF. Since a $n-n$ MSC is always both a mailbox and a $1-n$ MSC, it is clear that the cyclicity of \preceq_M implies that M is not $n-n$, because it means that we are not even able to find a linearization that is both mailbox and $1-n$. Moreover, since in a $n-n$ linearization the order in which messages are sent matches the order in which they are received, and unmatched send events can be executed only after matched send events, a $n-n$ MSC always has to satisfy the constraints imposed by the \triangleright_M relation. If \triangleright_M , then for sure there is no $n-n$ linearization for M . \square

Let the *Event Dependency Graph* (EDG) of a $n-n$ MSC M be a graph that has events as nodes and an edge between two events e_1 and e_2 if $e_1 \preceq_M e_2$. We now present an algorithm that, given the EDG of an $n-n$ MSC M , computes a $n-n$ linearization of M . We then show that, if \preceq_M is acyclic (i.e. it is a partial order), this algorithm always terminates correctly. This, along with Proposition C.14, effectively shows that Definition C.11 and Definition C.12 are equivalent.

Algorithm for finding a $n-n$ linearization. The input of this algorithm is the EDG of an MSC M , and it outputs a valid $n-n$ linearization for M , if M is $n-n$. The algorithm works as follows:

- (1) If there is a matched send event s with in-degree 0 in the EDG, add s to the linearization and remove it from the EDG, along with its outgoing edges, then jump to step 5. Otherwise, proceed to step 2.
- (2) If there are no matched send events in the EDG and there is an unmatched send event s with in-degree 0 in the EDG, add s to the linearization and remove it from the EDG, along with its outgoing edges, then jump to step 5. Otherwise, proceed to step 3.
- (3) If there is a receive event r with in-degree 0 in the EDG, such that r is the receive event of the first message whose send event was already added to the linearization, add r to the linearization and remove it from the EDG, along with its outgoing edges, then jump to step 5. Otherwise, proceed to step 4.
- (4) Throw an error and terminate.
- (5) If all the events of M were added to the linearization, return the linearization and terminate. Otherwise, go back to step 1.

We now need to show that (i) if this algorithm terminates correctly (i.e. step 4 is never executed), it returns a $n-n$ linearization, and (ii) if \preceq_M is acyclic, the algorithm always terminates correctly.

PROPOSITION C.15. *If the above algorithm returns a linearization for an MSC M , it is a $n-n$ linearization.*

PROOF. Note that, because of how step 2 works, the order (in the linearization) in which matched messages are sent is the same as the order in which they are received. Moreover, according to step 3, an unmatched send events is added to the linearization only if all the matched send events were already added. \square

PROPOSITION C.16. *Given an MSC M , the above algorithm always terminates correctly if \preceq_M is acyclic.*

PROOF. We want to prove that, if \preceq_M is acyclic, step 4 of the algorithm is never executed, i.e. it terminates correctly. Note that the acyclicity of \preceq_M implies that the EDG of M is a DAG. Moreover, at every step of the algorithm we remove nodes and edges from the EDG, so it still remains a DAG. The proof goes by induction on the number of events added to the linearization.

Base case: no event has been added to the linearization yet. Since the EDG is a DAG, there must be an event with in-degree 0. In particular, this has to be a send event (a receive event depends on its respective send event, so it cannot have in-degree 0). If it is a matched send event, step 1 is applied. If there are no matched send events, step 2 is applied on an unmatched send. We show

that it is impossible to have an unmatched send event of in-degree 0 if there are still matched send events in the EDG, so either step 1 or 2 are applied in the base case. Let s be one of those matched send events and let u be an unmatched send. Because of rule 4 in the definition of \bowtie_M , we have that $s \bowtie_M u$, which implies that u cannot have in-degree 0 if s is still in the EDG.

Inductive step: we want to show that we are never going to execute step 4. In particular, Step 4 is executed when none of the first three steps can be applied. This happens when there are no matched send events with in-degree 0 and one of the following holds:

- *There are still matched send events in the EDG with in-degree > 0 , there are no unmatched messages with in-degree 0, and there is no receive event r with in-degree 0 in the EDG, such that r is the receive event of the first message whose send event was already added to the linearization.* Since the EDG is a DAG, there must be at least one receive event with in-degree 0. We want to show that, between these receive events with in-degree 0, there is also the receive event r of the first message whose send event was added to the linearization, so that we can apply step 3 and step 4 is not executed. Suppose, by contradiction, that r has in-degree > 0 , so it depends on other events. For any maximal chain in the EDG that contains one of these events, consider the first event e , which clearly has in-degree 0. In particular, e cannot be a send event, because we would have applied step 1 or step 2. Hence, e can only be a receive event for a send event that was not the first added to the linearization (and whose respective receive still has not been added). However, this is also impossible, since $r_e \bowtie_M r$ implies $s_e \bowtie_M s$, and we could not have added s to the linearization before s_e . Because we got to a contradiction, the hypothesis that r has in-degree > 0 must be false, and we can indeed apply step 3.
- *There are still matched send events in the EDG with in-degree > 0 , there is at least one unmatched message with in-degree 0, and there is no receive event r with in-degree 0 in the EDG, such that r is the receive event of the first message whose send event was already added to the linearization.* We show that it is impossible to have an unmatched send event of in-degree 0 if there are still matched send events in the EDG. Let s be one of those matched send events and let u be an unmatched send. Because of rule 4 in the definition of \bowtie_M , we have that $s \bowtie_M u$, which implies that u cannot have in-degree 0 if s is still in the EDG.
- *There are no more matched send events in the EDG, there are no unmatched messages with in-degree 0, and there is no receive event r with in-degree 0 in the EDG, such that r is the receive event of the first message whose send event was already added to the linearization.* Very similar to the first case. Since the EDG is a DAG, there must be at least one receive event with in-degree 0. We want to show that, between these receive events with in-degree 0, there is also the receive event r of the first message whose send event was added to the linearization, so that we can apply step 3 and step 4 is not executed. Suppose, by contradiction, that r has in-degree > 0 , so it depends on other events. For any maximal chain in the EDG that contains one of these events, consider the first event e , which clearly has in-degree 0. In particular, e cannot be a send event, because by hypothesis there are no more send events with in-degree 0 in the EDG. Hence, e can only be a receive event for a send event that was not the first added to the linearization (and whose respective receive still has not been added). However, this is also impossible, since $r_e \bowtie_M r$ implies $s_e \bowtie_M s$, and we could not have added s to the linearization before s_e . Because we got to a contradiction, the hypothesis that r has in-degree > 0 must be false, and we can indeed apply step 3.

We showed that, if \bowtie_M is acyclic, the algorithm always terminates correctly and computes a valid n - n linearization. \square

Definition C.17 (RSC MSC). An MSC $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$ is a RSC MSC if it has no unmatched send events and there is a linearization \rightsquigarrow where any matched send event is immediately followed by its respective receive event.

Following the characterization given in [Charron-Bost et al. 1996, Theorem 4.4], we also give an alternative but equivalent definition of RSC MSC.

Definition C.18. Let M be an MSC. A crown of size k in M is a sequence $\langle (s_i, r_i), i \in \{1, \dots, k\} \rangle$ of pairs of corresponding send and receive events such that

$$s_1 <_M r_2, s_2 <_M r_3, \dots, s_{k-1} <_M r_k, s_k <_M r_1.$$

D: Specify somewhere that $<_M = (\rightarrow \cup \triangleleft)^+$.

Definition C.19 (RSC alternative). An MSC $M = (\mathcal{E}, \rightarrow, \triangleleft, \lambda)$ is a RSC MSC if and only if it does not contain any crown.

Proposition C.9 shows that $\text{MSC}_{1-n} \subset \text{MSC}_{\text{mb}} \subset \text{MSC}_{\text{co}} \subset \text{MSC}_{\text{p2p}} \subset \text{MSC}_{\text{asy}}$, as shown by Fig. 7.

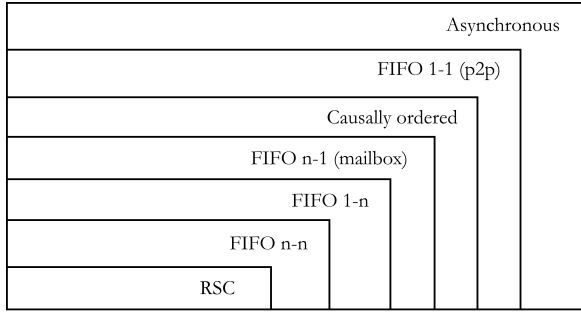


Fig. 7. The hierarchy of MSC classes.

C.2 Monadic Second-Order Logic

The set of MSO formulas over (asynchronous) MSCs (over \mathbb{P} and \mathbb{M}) is given by the grammar $\varphi ::= \text{true} \mid x \rightarrow y \mid x \triangleleft y \mid \lambda(x) = a \mid x = y \mid x \in X \mid \exists x. \varphi \mid \exists X. \varphi \mid \varphi \vee \varphi \mid \neg \varphi$, where $a \in \Sigma$, x and y are first-order variables, interpreted as events of an MSC, and X is a second-order variable, interpreted as a set of events. We assume that we have an infinite supply of variables, and we use common abbreviations such as \wedge , \Rightarrow , \forall , etc. The satisfaction relation is defined in the standard way and self-explanatory. For example, the formula $\neg \exists x. (\bigvee_{a \in \text{Send}(_, _, _)} \lambda(x) = a \wedge \neg \text{matched}(x))$ with $\text{matched}(x) = \exists y. x \triangleleft y$ says that there are no unmatched send events. It is not satisfied by MSC M_1 of Fig. ??, as message m_1 is not received, but by M_4 from Fig. ??.

Given a sentence φ , i.e., a formula without free variables, we let $L(\varphi)$ denote the set of MSCs that satisfy φ . Since we have defined the set of MSO formulas over asynchronous MSCs, the formula $\varphi_{\text{asy}} = \text{true}$ clearly describes the set of asynchronous MSCs, i.e. $L(\varphi_{\text{asy}}) = \text{MSC}_{\text{asy}}$. It is worth mentioning that the (reflexive) transitive closure of a binary relation defined by an MSO formula⁶ with free variables x and y , such as $x \rightarrow y$, is MSO-definable so that the logic can freely use formulas of the form $x \rightarrow^+ y$, $x \rightarrow^* y$ or $x \leq y$ (where \leq is interpreted as \leq_M for the given MSC M).

⁶See Section ?? for details.

Peer-to-peer MSCs. The set of p2p MSCs is MSO-definable as

$$\varphi_{p2p} = \neg \exists s. \exists s'. \left(\bigvee_{p \in \mathbb{P}, q \in \mathbb{P}} \bigvee_{a, b \in \text{Send}(p, q, _)} (\lambda(s) = a \wedge \lambda(s') = b) \wedge s \rightarrow^+ s' \wedge (\psi_1 \vee \psi_2) \right)$$

where ψ_1 and ψ_2 are

$$\psi_1 = \exists r. \exists r'. \left(\begin{array}{cc} s \triangleleft r & \wedge \\ s' \triangleleft r' & \wedge \\ r' \rightarrow^+ r & \end{array} \right) \quad \psi_2 = (\neg \text{matched}(s) \wedge \text{matched}(s'))$$

$$\text{matched}(x) = \exists y. x \triangleleft y$$

The property φ_{p2p} says that there cannot be two matched send events s and s' , with the same sender and receiver, such that either (i) $s \rightarrow^+ s'$ and their receipts happen in the reverse order, or (ii) s is unmatched and s' is matched. In other words, it ensures that channels operate in FIFO mode, where an unmatched messages blocks the receipt of all the subsequent messages on that channel. The set MSC_{p2p} is therefore MSO-definable as $\text{MSC}_{p2p} = L(\varphi_{p2p})$.

Causally ordered MSCs. Given an MSC M , it is causally ordered if and only if it satisfies the MSO formula

$$\varphi_{co} = \neg \exists s. \exists s'. \left(\bigvee_{q \in \mathbb{P}} \bigvee_{a, b \in \text{Send}(_, q, _)} (\lambda(s) = a \wedge \lambda(s') = b) \wedge s \leq_M s' \wedge (\psi_1 \vee \psi_2) \right)$$

where ψ_1 and ψ_2 are the same formulas used for p2p.

The property φ_{co} says that there cannot be two send events s and s' , with the same recipient, such that $s \leq_M s'$ and either (i) their corresponding receive events r and r' happen in the opposite order, i.e. $r' \rightarrow^+ r$, or (ii) s is unmatched and s' is matched. The set MSC_{co} of causally ordered MSCs is therefore MSO-definable as $\text{MSC}_{co} = L(\varphi_{co})$.

Mailbox MSCs. Given an MSC M , it is a mailbox MSC if and only if it satisfies the MSO formula

$$\varphi_{mb} = \varphi_{p2p} \wedge \neg \exists x. \exists y. (\neg(x = y) \wedge x \leq_M y \wedge y \leq_M x)$$

The set MSC_{mb} of mailbox MSCs is therefore MSO-definable as $\text{MSC}_{mb} = L(\varphi_{mb})$.

D: This does not match my definition of mailbox MSC, should I also give the alternative definition (the one in the concur paper) or rewrite everything according to my definition? Give both definitions and prove that they are equivalent

1- n MSCs. Following Definition C.8, an MSC M is a 1- n MSC if and only if it satisfies the MSO formula

$$\varphi_{1-n} = \neg \exists x. \exists y. (\neg(x = y) \wedge x \leq_M y \wedge y \leq_M x)$$

Recall that \leq_M is the union of the MSO-definable relations \rightarrow , \triangleleft , and \triangleleft_M . In particular, we can define $x \triangleleft_M y$ as

$$x \triangleleft_M y = \left(\bigvee_{\substack{p \in \mathbb{P} \\ a, b \in \text{Send}(p, _, _)}} (\lambda(x) = a \wedge \lambda(y) = b) \wedge \text{matched}(x) \wedge \neg \text{matched}(y) \right) \vee \left(\bigvee_{\substack{p \in \mathbb{P} \\ a, b \in \text{Rec}(p, _, _)}} (\lambda(x) = a \wedge \lambda(y) = b) \wedge \exists x'. \exists y'. (x' \triangleleft x \wedge y' \triangleleft y \wedge x' \rightarrow^+ y') \right)$$

The MSO formula for $x \triangleleft_M y$ closely follows Definition C.8. The set MSC_{1-n} of 1- n MSCs is therefore MSO-definable as $\text{MSC}_{1-n} = L(\varphi_{1-n})$.

n - n MSCs. Following Definition C.12, an MSC M is a n - n MSC if and only if it satisfies the MSO formula

$$\varphi_{n-n} = \neg \exists x. \exists y. (\neg(x = y) \wedge x \bowtie_M y \wedge y \bowtie_M x)$$

In particular, we can define $x \bowtie_M y$ as

$$x \bowtie_M y = \left(\bigvee_{a,b \in \text{Send}(_, _, _)} (\lambda(x) = a \wedge \lambda(y) = b) \wedge \text{matched}(x) \wedge \neg \text{matched}(y) \right) \vee (x \bowtie_M y) \vee \psi_3 \vee \psi_4$$

where ψ_3 and ψ_4 are defined as

$$\begin{aligned} \psi_3 &= \bigvee_{a,b \in \text{Rec}(_, _, _)} (\lambda(x) = a \wedge \lambda(y) = b) \wedge \exists x'. \exists y'. (x' \triangleleft x \wedge y' \triangleleft y) \wedge (x' \bowtie_M y') \wedge \neg(x \bowtie_M y) \\ \psi_4 &= \bigvee_{a,b \in \text{Send}(_, _, _)} (\lambda(x) = a \wedge \lambda(y) = b) \wedge \exists x'. \exists y'. (x \triangleleft x' \wedge y \triangleleft y') \wedge (x' \bowtie_M y') \wedge \neg(x \bowtie_M y) \end{aligned}$$

The MSO formula for $x \bowtie_M y$ closely follows Definition C.12. The set MSC_{n-n} of n - n MSCs is therefore MSO-definable as $\text{MSC}_{n-n} = L(\varphi_{n-n})$.

RSC MSCs. Following Definition C.19, an MSC M is a RSC MSC if and only if it satisfies the MSO formula

$$\Phi_{\text{RSC}} = \neg \exists s_1. \exists s_2. s_1 \alpha s_2 \wedge s_2 \alpha^* s_1$$

where α is defined as

$$s_1 \alpha s_2 = \bigvee_{e \in \text{Send}(_, _, _)} (\lambda(s_1) = e) \wedge s_1 \neq s_2 \wedge \exists r_2. (s_1 < r_2 \wedge s_2 \triangleleft r_2)$$

REFERENCES

Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Étienne Lozes, and Amrita Suresh. 2021. A Unifying Framework for Deciding Synchronizability. In *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference (LIPIcs, Vol. 203)*, Serge Haddad and Daniele Varacca (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:18. <https://doi.org/10.4230/LIPIcs.CONCUR.2021.14>

Bernadette Charron-Bost, Friedemann Mattern, and Gerard Tel. 1996. Synchronous, Asynchronous, and Causally Ordered Communication. *Distributed Comput.* 9, 4 (1996), 173–191. <https://doi.org/10.1007/s004460050018>

Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. 2016. On the diversity of asynchronous communication. *Formal Aspects Comput.* 28, 5 (2016), 847–879. <https://doi.org/10.1007/s00165-016-0379-x>