



Guessing the Buffer Bound for k-Synchronizability

Cinzia Di Giusto, Laetitia Laversa, Etienne Lozes

► To cite this version:

Cinzia Di Giusto, Laetitia Laversa, Etienne Lozes. Guessing the Buffer Bound for k-Synchronizability. International Journal of Foundations of Computer Science, 2022, 34 (8), pp.1051-1076. 10.1142/S0129054122430018 . hal-03890605

HAL Id: hal-03890605

<https://hal.science/hal-03890605>

Submitted on 7 Mar 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Guessing the buffer bound for k -synchronizability

Cinzia Di Giusto¹ , Laetitia Laversa¹ , and Etienne Lozes¹ 

Université Côte d’Azur, CNRS, I3S, Sophia Antipolis, France

Abstract. A communicating system is k -synchronizable if all of the message sequence charts representing the executions can be divided into slices of k sends followed by k receptions. It was previously shown that, for a fixed given k , one could decide whether a communicating system is k -synchronizable. This result is interesting because the reachability problem can be solved for k -synchronizable systems. However, the decision procedure assumes that the bound k is fixed. In this paper we improve this result and show that it is possible to decide if such a bound k exists.

Keywords: communicating automata · MSC · synchronizability

1 Introduction

Communicating finite state machines [4] model distributed systems where participants exchange messages via FIFO buffers. Due to the unboundedness of the buffers, the model is Turing powerful as soon as there are two participants and two queues. In order to recover decidability, several works introduced restrictions on the model, for instance: lossiness of the channels [1], specific topologies, or bounded context switching [13]. Another line of research focused on analyzing the system under the assumption that the semantics is synchronous [2,9,14,8,6,5,12,11] or that buffers are bounded. This assumption is not as restrictive as it may seem at first, because several systems enjoy the property that their execution, although not necessarily bounded, can be simulated by a causally equivalent bounded execution. Existentially k -bounded communicating systems [10] are precisely the systems whose message sequence charts can be generated by k -bounded executions. In particular, the reachability problem is decidable for existentially k -bounded communicating systems. A limitation of this framework is that the bound k on the buffer size must be fixed. A natural question is whether the existence of such a bound can be decided. Genest, Kuske and Muscholl answered this question negatively [10]. Bouajjani et al. [3]¹ introduced a variant of existentially k -bounded communicating systems they called k -synchronizable systems. A system is k -synchronizable if each of its execution is causally equivalent to a sequence of communication rounds composed of at most k sends followed by at most k receptions. In particular, each execution of a k -synchronizable system is causally equivalent to a k -bounded execution (provided

¹ The results in [3] have then been refined in [7]

all messages are eventually received). Like for existentially bounded systems, the reachability problem becomes decidable for k -synchronizable systems, and the membership problem - whether a given system is k -synchronizable for a fixed given k is decidable as well. Bouajjani et al. conjectured that the existence of a bound k on the size of the communication rounds was undecidable.

Instead, in this paper, we show that this problem is decidable. This result contrasts with the negative result about the same question for existentially bounded communicating systems. There is an important difference between existentially bounded and k -synchronizable ones that explains this situation. Existentially bounded systems deal with peer-to-peer communications, with one buffer per pair of machines, whereas k -synchronizable systems deal with mailbox communications where one buffer per machine merges all incoming messages.

The paper is organized as follows: in the next section, we introduce preliminary definitions on communicating automata and k -synchronizable systems. In Section 3 we explain the general strategy for computing the bound k , which is to compute the automata of two regular languages: the language of reachable exchanges, and the language of prime exchanges. In Section 4 we focus on reachable exchanges, and in Section 5 on prime exchanges. Section 6 lastly computes the bound k . Finally Section 7 concludes with some final remarks. An appendix with additional material and proofs is added for the reviewer convenience.

2 Preliminaries

Let \mathbb{V} be a finite set of messages and \mathbb{P} a finite set of processes exchanging messages. A send action, denoted $send(p, q, v)$, designates the sending of message v from process p to process q , storing it in the queue of q . Similarly, a receive action $rec(q, v)$ expresses that process q pops message v from its queue of incoming messages. We write a to denote a send or receive action. Let $S = \{send(p, q, v) \mid p, q \in \mathbb{P}, v \in \mathbb{V}\}$ be the set of send actions and $R = \{rec(q, v) \mid q \in \mathbb{P}, v \in \mathbb{V}\}$ the set of receive actions. S_p and R_p stand for the set of sends and receives of process p respectively.

A *system* is a tuple $\mathfrak{S} = ((L_p, \delta_p, l_p^0) \mid p \in \mathbb{P})$ where, for each process p , L_p is a finite set of local control states, $\delta_p \subseteq (L_p \times (S_p \cup R_p) \times L_p)$ is the transition relation and l_p^0 is the initial state. In the rest of the paper, when talking about a system \mathfrak{S} , we may also identify it with the global automaton obtained as the product of the process automata and denoted $(L_{\mathfrak{S}}, \delta_{\mathfrak{S}}, \mathbf{l}_0)$ where $L_{\mathfrak{S}} = \prod_{p \in \mathbb{P}} L_p$ is the set of global control states, $\mathbf{l}_0 = (l_p^0)_{p \in \mathbb{P}}$ is the initial global control state and $((l_1, \dots, l_q, \dots, l_n), a, (l_1, \dots, l'_q, \dots, l_n)) \in \delta_{\mathfrak{S}}$ iff $(l_q, a, l'_q) \in \delta_q$ for $q \in \mathbb{P}$. We write \mathbf{l} in bold to denote the tuple of control states $(l_p)_{p \in \mathbb{P}}$, and we sometimes write $l_q \xrightarrow{a}_q l'_q$ (resp. $\mathbf{l} \xrightarrow{a} \mathbf{l}'$) for $(l_q, a, l'_q) \in \delta_q$ (resp. $(\mathbf{l}, a, \mathbf{l}') \in \delta_{\mathfrak{S}}$). We write $\xrightarrow{a_1 \dots a_n}$ for $\xrightarrow{a_1} \dots \xrightarrow{a_n}$.

A *configuration* is a pair (\mathbf{l}, Buf) where $\mathbf{l} = (l_p)_{p \in \mathbb{P}} \in L_{\mathfrak{S}}$ is a global control state of \mathfrak{S} , and $\text{Buf} = (b_p)_{p \in \mathbb{P}} \in (\mathbb{V}^*)^{\mathbb{P}}$ is a vector of buffers, each b_p being a word over \mathbb{V} . Buf_0 stands for the vector of empty buffers. The mailbox semantics of a system is defined by the two rules below.

$$\begin{array}{c}
 \text{[SEND]} \\
 \frac{\mathbf{l} \xrightarrow{\text{send}(p,q,v)} \mathbf{l}' \quad b'_q = b_q \cdot v}{(\mathbf{l}, \text{Buf}) \xrightarrow{\text{send}(p,q,v)} (\mathbf{l}', \text{Buf}[b'_q/b_q])}
 \end{array}
 \qquad
 \begin{array}{c}
 \text{[RECEIVE]} \\
 \frac{\mathbf{l} \xrightarrow{\text{rec}(q,v)} \mathbf{l}' \quad b_q = v \cdot b'_q}{(\mathbf{l}, \text{Buf}) \xrightarrow{\text{rec}(q,v)} (\mathbf{l}', \text{Buf}[b'_q/b_q])}
 \end{array}$$

In this paper, we focus on **mailbox** semantics. An execution $e = a_1 \cdots a_n$ is a sequence of actions in $S \cup R$ such that $(\mathbf{l}_0, \text{Buf}_0) \xrightarrow{a_1} \cdots \xrightarrow{a_n} (\mathbf{l}, \text{Buf})$ for some \mathbf{l} and Buf . As usual, \xRightarrow{e} stands for $\xrightarrow{a_1} \cdots \xrightarrow{a_n}$. We write $\text{asEx}(\mathfrak{S})$ to denote the set of executions of a system \mathfrak{S} . Executions impose a total order over the actions. To stress the causal dependencies between messages we use message sequence charts (MSCs) that only impose an order between matched pairs of actions and between the actions of a same process.

Definition 1 (Message Sequence Chart). A message sequence chart μ is a tuple $(Ev, \lambda, \prec_{po}, \prec_{src})$ such that

1. Ev is a finite set of events partially ordered under $(\prec_{po} \cup \prec_{src})^*$,
2. $\lambda : Ev \rightarrow S \cup R$ tags each event with an action,
3. for each process p , \prec_{po} induces a total order on the events of p , i.e. on $\lambda^{-1}(S_p \cup R_p)$,
4. (Ev, \prec_{src}) is the graph of a bijection between a subset of $\lambda^{-1}(S)$ and the whole of $\lambda^{-1}(R)$
5. for all $s \prec_{src} r$, there are p, q, v such that $\lambda(s) = \text{send}(p, q, v)$ and $\lambda(r) = \text{rec}(q, v)$.

Definition 2 (Concatenation of MSCs). Let $\mu_1 = (Ev_1, \lambda_1, \prec_{po}^1, \prec_{src}^1)$ and $\mu_2 = (Ev_2, \lambda_2, \prec_{po}^2, \prec_{src}^2)$ be two MSCs. Their concatenation $\mu_1 \cdot \mu_2$ is the MSC $\mu = (Ev, \lambda, \prec_{po}, \prec_{src})$ such that:

- $Ev = Ev_1 \cup Ev_2$
- $\lambda = \lambda_1 \cup \lambda_2$
- $\prec_{po} = \prec_{po}^1 \cup \prec_{po}^2 \cup \bigcup_{p \in \mathbb{P}} \{(e_1, e_2) \mid e_1 \in \lambda_1^{-1}(S_p \cup R_p), e_2 \in \lambda_2^{-1}(S_p \cup R_p)\}$
- $\prec_{src} = \prec_{src}^1 \cup \prec_{src}^2$.

In a sequence of actions $e = a_1 \cdots a_n$, a send action $a_i = \text{send}(p, q, v)$ is *matched* by a reception $a_j = \text{rec}(q', v')$ (denoted by $a_i \vdash a_j$) if $i < j$, $p = p'$, $q = q'$, $v = v'$, and there is $\ell \geq 1$ such that a_i and a_j are the ℓ th actions of e with these properties respectively. A send action a_i is *unmatched* if there is no matching reception in e .

The MSC associated with the execution $e = a_1 \cdots a_n$ is $(Ev, \lambda, \prec_{po}, \prec_{src})$ where $Ev = \{1, \dots, n\}$, $\lambda(i) = a_i$, $i \prec_{po} j$ iff $i < j$ and $\{a_i, a_j\} \subseteq S_p \cup R_p$ for some p , and $i \prec_{src} j$ if $a_i \vdash a_j$.

When v is either an unmatched $\text{send}(p, q, v)$ or a pair of matched actions $\{\text{send}(p, q, v), \text{rec}(q, v)\}$, we write $\text{proc}_S(v)$ for p and $\text{proc}_R(v)$ for q . Note that $\text{proc}_R(v)$ is defined even if v is unmatched. An MSC is depicted with vertical timelines (one for each process) where time goes from top to bottom. Points on the lines represent events of this process. We draw an arc between two matched events and a dashed arc to depict an unmatched send. The concatenation $\mu_1 \cdot \mu_2$ of two MSCs is the union of the two MSCs where, for each p , all p -events of

μ_1 are considered \prec_{po} smaller than all p -events of μ_2 . We write $msc(e)$ for the MSC associated with the execution e , and we say that a sequence of actions e is a linearization of a given MSC if it is the sequence of actions induced by a total order extending $(\prec_{po} \cup \prec_{src})^*$. We write $asTr(\mathfrak{S})$ for the set $\{msc(e) \mid e \in asEx(\mathfrak{S})\}$. We write $1 \xrightarrow{\mu} 1'$ to denote that $1 \xRightarrow{e} 1'$ for any linearization e of μ . Finally, we recall from [7] the definition of causal delivery that allows to consider only MSCs that correspond to executions in the mailbox semantics.

Definition 3 (Causal delivery). Let $\mu = (Ev, \lambda, \prec_{po}, \prec_{src})$ be an MSC. We say that μ satisfies causal delivery if it admits a linearization with the total order $<$ such that for any two events $s_1, s_2 \in Ev$, if $s_1 < s_2$, $\lambda(s_1) = send(p, q, v)$ and $\lambda(s_2) = send(p', q, v')$ for a same destination process q , then either s_2 is unmatched, or there are r_1, r_2 such that $s_1 \prec_{src} r_1$, $s_2 \prec_{src} r_2$, and $r_1 < r_2$.

A k -exchange (with $k \geq 1$) is an MSC that admits a linearization $e \in S^{\leq k} R^{\leq k}$ starting with at most k sends and followed by at most k receives. An MSC is k -synchronous if it can be chopped into a sequence of k -exchanges.

Definition 4 (k -synchronous). An MSC μ is k -synchronous if $\mu = \mu_1 \cdot \mu_2 \cdots \mu_n$ where, for all $i \in [1..n]$, μ_i is a k -exchange.

For instance, the MSC μ_1 depicted on Fig. 1 is 2-synchronous, as it can be split in two 2-exchanges.

An execution e is k -synchronizable if $msc(e)$ is k -synchronous. A system \mathfrak{S} is k -synchronizable if all its executions are k -synchronizable.

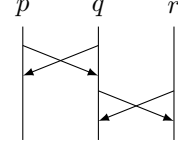


Fig. 1. MSC μ_1

Theorem 1 ([3,7]). It is decidable whether a system \mathfrak{S} is k -synchronizable for a given k . Moreover, it is decidable to know whether a control state is reachable under the assumption that \mathfrak{S} is k -synchronizable.

This result is interesting but somehow incomplete as it assumes that a fixed value of the parameter k has been found. We aim at answering this limitation by computing the *synchronizability degree* of a given system.

Definition 5 (Synchronizability degree). The synchronizability degree $sd(\mathfrak{S})$ of a system \mathfrak{S} is the smallest k such that \mathfrak{S} is k -synchronizable. In particular, $sd(\mathfrak{S}) = \infty$ if \mathfrak{S} is not k -synchronizable for any k .

3 Largest prime reachable exchange

In this section, we relate the synchronizability degree of a system to the size of a “maximal, prime, reachable exchange”. We start with defining these notions.

An *exchange* is a k -exchange for some arbitrary k , and we call k the size of the exchange. An exchange μ is *reachable* if there exist exchanges μ_1, \dots, μ_n for

some $n \geq 0$ and such that $\mu_1 \cdots \mu_n \cdot \mu \in asTr(\mathfrak{S})$. An exchange μ is *prime* if there does not exist a decomposition $\mu = \mu_1 \cdot \mu_2$ in two non-empty exchanges. For instance, the 2-exchange (depicted by the MSC μ_2 , Fig. 2) with linearization:

$$send(p, q, v_1) \cdot send(r, q, v_2) \cdot rec(q, v_1) \cdot rec(q, v_2)$$

is not prime, as it can be factored in two 1-exchanges as follows

$$send(p, q, v_1) \cdot rec(q, v_1) \quad \cdot \quad send(r, q, v_2) \cdot rec(q, v_2).$$

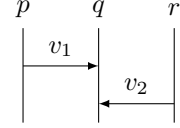


Fig. 2. MSC μ_2

The size of the biggest prime reachable exchange is related to the synchronizability degree $sd(\mathfrak{S})$ by the following property.

Lemma 1. *Let $k \in \mathbb{N} \cup \{\infty\}$ be the supremum of the sizes of all prime reachable exchanges. (1) If $k = \infty$, then $sd(\mathfrak{S}) = \infty$ (2) if $k < \infty$, then either \mathfrak{S} is k -synchronizable and $sd(\mathfrak{S}) = k$, or \mathfrak{S} is not k -synchronizable and $sd(\mathfrak{S}) = \infty$.*

Proof. Let $k \in \mathbb{N} \cup \{\infty\}$ be the supremum of the sizes of all prime reachable exchanges.

Assume that there exists K such that \mathfrak{S} is K -synchronizable. Let us show that $k \leq K$ and \mathfrak{S} is k -synchronizable.

- $k \leq K$. Assume by contradiction that $k \geq K + 1$. Then there exists exchanges μ_1, \dots, μ_n, μ such that $\mu_1 \cdots \mu_n \cdot \mu \in asTr(\mathfrak{S})$ and μ is prime of size $K + 1$. Since μ is prime, it corresponds to a strongly connected component of size $K + 1$ of the conflict graph of $\mu_1 \cdots \mu_n \cdot \mu$, so $\mu_1 \cdots \mu_n \cdot \mu$ cannot be K -synchronous: contradiction.
- \mathfrak{S} is k -synchronizable. Let $\mu \in asTr(\mathfrak{S})$ be fixed and let us show that it can be chopped into a sequence of k exchanges. Since by hypothesis \mathfrak{S} is K -synchronizable, there are K -exchanges μ_1, \dots, μ_n such that $\mu = \mu_1 \cdots \mu_n$. Up to decomposing each μ_i as a product of prime exchanges, we can assume that all μ_i are prime. Moreover, they are all reachable, so their size is bounded by k . As a consequence, μ can be decomposed in a sequence of k -exchanges. \square

Since by Theorem 1 it is decidable whether \mathfrak{S} is k -synchronizable, it is enough to know k in order to compute $sd(\mathfrak{S})$. In order to compute k , we have to address two problems: the number of exchanges is possibly infinite, and one should examine sequences of arbitrarily many exchanges. To solve these issues, we are going to reduce to a problem on regular languages. Let $\Sigma = \{!?, !\} \times \mathbb{V} \times \mathbb{P}^2$; for better readability, we write $!v^{p \rightarrow q}$ (resp. $!v^{p \rightarrow q}$) for a Σ -symbol. To every Σ -word w we associate an MSC $msc(w)$ as follows. Consider the substitutions $\sigma_1 : \Sigma \rightarrow S$ and $\sigma_2 : \Sigma \rightarrow R \cup \{\epsilon\}$ such that $\sigma_1(!v^{p \rightarrow q}) = \sigma_1(!v^{p \rightarrow q}) = send(p, q, v)$, $\sigma_2(!v^{p \rightarrow q}) = rec(q, v)$ and $\sigma_2(!v^{p \rightarrow q}) = \epsilon$. Then $msc(w)$ is defined as $msc(\sigma_1(w)\sigma_2(w))$. Clearly, it is an exchange (by construction, it admits a linearization in S^*R^*), but more remarkably any reachable exchange can be represented by such a word.

Lemma 2. *For all reachable exchanges μ , there exists $w \in \Sigma^*$ s.t. $\mu = msc(w)$.*

Proof. Let μ be a reachable exchange, and let μ_1, \dots, μ_n be such that $\mu_1 \cdot \mu_2 \cdots \mu_n \cdot \mu \in asTr(\mathfrak{S})$. There is a linearization of $\mu_1 \cdots \mu_n \cdot \mu$ which follows the mailbox semantics. This linearization induces a linearization $lin(\mu)$ of μ that also follows the mailbox semantics. Then $lin(\mu)$ induces an enumeration $send(p_1, q_1, v_1), \dots, send(p_n, q_n, v_n)$ of the send events of μ . Let $w = a_1 \dots a_n$ where a_i is either $!v_i^{p_i \rightarrow q_i}$ if $send(p_i, q_i, v_i)$ is matched in μ , or $!v_i^{p_i \rightarrow q_i}$ if it is unmatched. Then, the claim is that $msc(w) = \mu$, or in other words, $\sigma_1(w)\sigma_2(w)$ is a linearization of μ . By contradiction, assume it is not. Then there are two events e, e' such that $e < e'$ in the enumeration $\sigma_1(w)\sigma_2(w)$ but $(e', e) \in (\prec_{po} \cup \prec_{src})^*$.

- if e, e' are two send events then e occurs before e' in $\sigma_1(w)$, i.e. e occurs before e' in $lin(\mu)$, which is a linearization of μ , and the contradiction with $(e', e) \in (\prec_{po} \cup \prec_{src})^*$.
- if e is a send event and e' a receive event, then $(e', e) \in (\prec_{po} \cup \prec_{src})^*$ contradicts the fact that μ is an exchange.
- if e is a receive event and e' is a send event, then $e < e'$ wrt $\sigma_1(w)\sigma_2(w)$ contradicts the definition of σ_1, σ_2 .
- assume finally that e and e' are receive events. From $(e', e) \in (\prec_{po} \cup \prec_{src})^*$, we deduce that $e' \prec_{po} e$, because μ is an exchange. Let s, s' be the matching send events of e, e' respectively. Since $e < e'$ wrt $\sigma_1(w)\sigma_2(w)$, $s < s'$ wrt $\sigma_1(w)\sigma_2(w)$, and therefore $s < s'$ wrt $lin(\mu)$. But $e' < e$ wrt $lin(\mu)$ because $e' \prec_{po} e$, which violates the mailbox semantics: contradiction.

□

The proof follows from the fact that it is always possible to receive messages in the same global order as they have been sent. Such a property would not hold for peer-to-peer communications, as we can see in the following counter-example.

Consider MSC μ_6 on the right. This MSC does not satisfy causal delivery in a mailbox semantics, because the sending of v_1 happens before the sending of v_4 , and the reception of v_4 happens before the reception of v_1 . For this reason, there is no word w such that $msc(w)$ corresponds to this MSC: such a word would give a linearization that would correspond to a valid mailbox execution. On the other hand, this MSC satisfies causal delivery in a peer-to-peer semantics. For instance, the following linearization is a peer-to-peer execution:

$$!v_3 \cdot !v_4 \cdot !v_1 \cdot ?v_2 \cdot ?v_2 \cdot ?v_3 \cdot ?v_4 \cdot ?v_1$$

We can now define two languages over Σ :

$$\mathcal{L}_r = \{w \in \Sigma^* \mid msc(w) \text{ is reachable}\} \text{ and } \mathcal{L}_p = \{w \in \Sigma^* \mid msc(w) \text{ is prime}\}$$

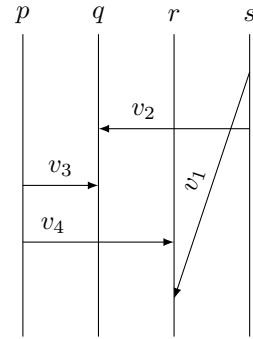


Fig. 3. MSC μ_6

Then the bound k we are looking for is the length of the longest word in $\mathcal{L}_r \cap \mathcal{L}_p$. It suffices to show that both \mathcal{L}_r and \mathcal{L}_p are effective regular languages to get an algorithm for computing k . This is the content of Sections 4 and 5.

4 Regularity of reachable exchanges

In this section, we aim at defining a finite state automaton that accepts a word $w \in \Sigma^*$ iff $msc(w)$ is reachable, that is, iff there exists $\mu_1, \mu_2, \dots, \mu_n$ such that $\mu_1 \cdot \mu_2 \cdots \mu_n \cdot msc(w) \in asTr(\mathfrak{S})$. Now, observe that the prefix $\mu_1 \cdot \mu_2 \cdots \mu_n$ brings the system in a certain global control state that conditions what can be done by $msc(w)$. Moreover, the presence of unmatched messages in a buffer imposes that none of the subsequent messages sent to the same buffer can be read.

The construction of the automaton accepting \mathcal{L}_r proceeds in three separate steps. First, we build an automaton that accepts the language of all words that code an exchange, starting in a certain global control state **in**, and ending in another global control state **fin**, and possibly not satisfying causal delivery. Secondly, we consider the set of MSCs that satisfy causal delivery. We define automata that recognize the words coding MSCs starting from a certain “buffer state” and ending in another “buffer state”, the “buffer state” characterizing whether or not the MSC satisfies causal delivery. Finally, we show that \mathcal{L}_r is a boolean combination of the languages of some of these automata.

4.1 Automata of the control states

We consider triples of global states (**in**, **mid**, **fin**), representing the exchanges such that **mid** can be reached only with sends from **in** and **fin** can be reached only with receptions from **mid**. We want to define an automaton $SR(\mathbf{in}, \mathbf{mid}, \mathbf{fin})$ that recognizes the words coding such exchanges. Intuitively, $SR(\mathbf{in}, \mathbf{mid}, \mathbf{fin})$ is a product of on the one hand the global automaton \mathfrak{S} restricted to send transitions and on the other hand \mathfrak{S} restricted to receive transitions. For each send action, either the reception is available and a matched message possible, or there is no corresponding reception and so we obtain an unmatched message.

Definition 6 (Automaton of control states). *Let \mathfrak{S} be a system and **in**, **mid**, **fin** global states. $SR(\mathbf{in}, \mathbf{mid}, \mathbf{fin}) = (L_{SR}, \delta_{SR}, \mathbf{l}_{SR}^0, F_{SR})$ is the automaton where:*

- $L_{SR} = \{(\mathbf{l}, \mathbf{l}') \mid \mathbf{l}, \mathbf{l}' \in L_{\mathfrak{S}}\}; \mathbf{l}_{SR}^0 = (\mathbf{in}, \mathbf{mid}); F_{SR} = \{(\mathbf{mid}, \mathbf{fin})\};$
- for each $(\mathbf{l}_s, send(p, q, v), \mathbf{l}'_s) \in \delta_{\mathfrak{S}}:$
 - $((\mathbf{l}_s, \mathbf{l}), !v^{p \rightarrow q}, (\mathbf{l}'_s, \mathbf{l})) \in \delta_{SR}$ for $\mathbf{l} \in L_{\mathfrak{S}};$
 - if $(\mathbf{l}_r, rec(q, v), \mathbf{l}'_r) \in \delta_{\mathfrak{S}}$ then $((\mathbf{l}_s, \mathbf{l}_r), !?v^{p \rightarrow q}, (\mathbf{l}'_s, \mathbf{l}'_r)) \in \delta_{SR}$

We denote $\mathcal{L}(SR(\mathbf{in}, \mathbf{mid}, \mathbf{fin}))$ the language of a such automaton. This is an example of the construction.

Example 1. Let \mathfrak{S}_1 be the system whose process automata p, q and r are depicted in Fig 4. For the triple (**in**, **mid**, **fin**) where **in** = (0, 0, 0), **mid** = (2, 0, 1) and

fin = (2, 1, 2), automaton **SR(in, mid, fin)** is depicted below the system and has for language:

$$\begin{aligned} \mathcal{L}(\mathbf{SR}(\mathbf{in}, \mathbf{mid}, \mathbf{fin})) = & \text{!?}a^{p \rightarrow r}(\text{!}c^{p \rightarrow q}\text{!?}b^{r \rightarrow q} + \text{!?}b^{r \rightarrow q}\text{!}c^{p \rightarrow q}) \\ & + \text{!?}b^{r \rightarrow q}\text{!?}a^{p \rightarrow r}\text{!}c^{p \rightarrow q} \end{aligned}$$

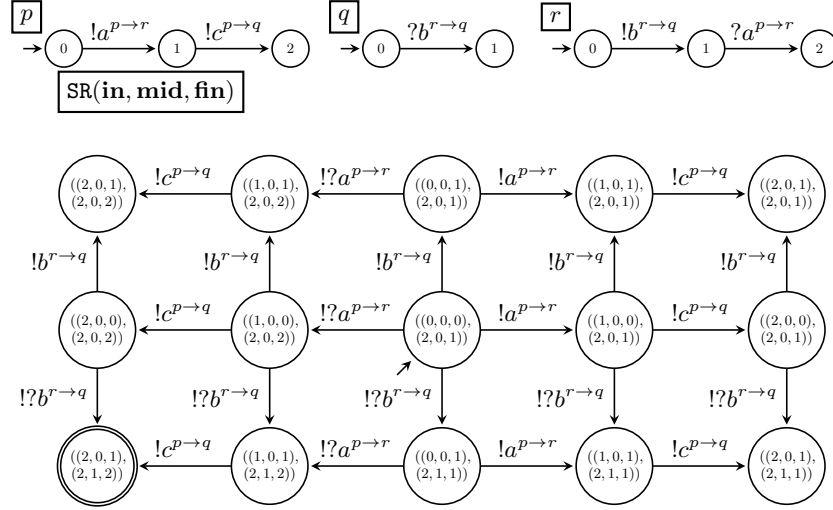


Fig. 4. System \mathfrak{S}_1 and Automaton **SR(in, mid, fin)**

Lemma 3. $w \in \mathcal{L}(\mathbf{SR}(\mathbf{in}, \mathbf{mid}, \mathbf{fin}))$ for some **mid** iff $\mathbf{in} \xrightarrow{msc(w)} \mathbf{fin}$.

Proof. Observe that, by construction of **SR(in, mid, fin)**, $\mathbf{l}_{\mathbf{SR}}^0 \xrightarrow{w} (\mathbf{l}, \mathbf{l}')$ iff $\mathbf{in} \xrightarrow{\sigma_1(w)} \mathbf{l}$ and $\mathbf{mid} \xrightarrow{\sigma_2(w)} \mathbf{l}'$ (this can be shown by an easy induction on the length of w). In particular, w is accepted iff $\mathbf{in} \xrightarrow{\sigma_1(w)} \mathbf{mid}$ and $\mathbf{mid} \xrightarrow{\sigma_2(w)} \mathbf{fin}$, which is equivalent to $\mathbf{in} \xrightarrow{msc(w)} \mathbf{fin}$. \square

4.2 Automata of causal delivery exchanges

Let us now move to the trickier part, namely the recognition of words coding MSCs that satisfy causal delivery. Let $\mu = (Ev, \lambda, \prec_{po}, \prec_{src})$ be an MSC, and $v \in \lambda^{-1}(S)$ a send event, we write $ev_S(v)$ for the event v and, when it exists, $ev_R(v)$ for the event $v' \in \lambda^{-1}(R)$ such that $v \prec_{src} v'$. We say that v is unmatched

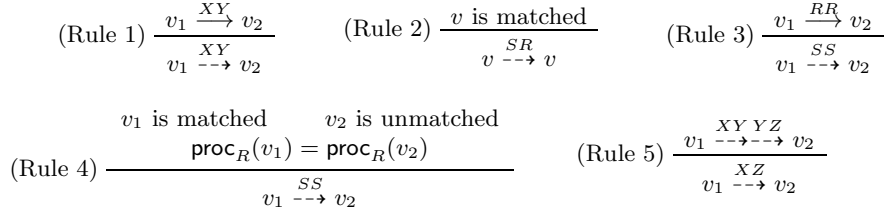


Fig. 6. Deduction rules for extended dependency edges of the conflict graph

if $ev_R(v)$ is undefined. We recall from [3] the notion of conflict graph. Intuitively, it captures some (but not all) causal dependencies between events. The figure on the right represents an MSC and its associated conflict graph.

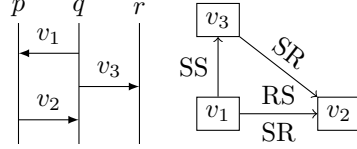


Fig. 5. MSC μ_3 and its conflict graph

Definition 7 (Conflict Graph). The conflict graph $\text{CG}(\mu)$ of an MSC $\mu = (Ev, \lambda, \prec_{po}, \prec_{src})$ is the labeled graph $(V, \{\xrightarrow{XY}\}_{X,Y \in \{R,S\}})$ where $V = \lambda^{-1}(S)$, and for all $v, v' \in V$, there is a XY dependency edge $v \xrightarrow{XY} v'$ between v and v' ($X, Y \in \{S, R\}$), if $ev_X(v)$ and $ev_Y(v')$ are defined and $ev_X(v) \prec_{po} ev_Y(v')$.

The extended conflict graph [7] $\text{ECG}(\mu)$ is obtained by adding all dashed edges $v \dashrightarrow v'$ satisfying the relation \dashrightarrow in Fig. 6. Intuitively, $v \dashrightarrow v'$ expresses that the event X of v must happen before the event Y of v' due to: their order on the same machine (Rule 1), or the fact that a send happens before its matching receive (Rule 2), or to the mailbox semantics (Rules 3 and 4), or because of a chain of such dependencies (Rule 5). This captures all constraints induced by the mailbox communication, and it has been shown that an MSC satisfies causal delivery if and only if its extended conflict graph is acyclic (Theorem 2 in [7]).

We build an automaton that recognizes the words w such that $\text{msc}(w)$ satisfies causal delivery. To this aim, we associate to each MSC a “buffer state” that contains enough information to determine whether its extended conflict graph is acyclic. We write \mathbb{B} for the set $(2^{\mathbb{P}} \times 2^{\mathbb{P}})^{\mathbb{P}}$. The buffer state $\mathcal{B}(\mu) \in \mathbb{B}$ of the MSC μ is the tuple $\mathcal{B}(\mu) = (\mathcal{C}_{S,p}^\mu, \mathcal{C}_{R,p}^\mu)_{p \in \mathbb{P}}$ such that for all $p \in \mathbb{P}$:

$$\begin{aligned}
 \mathcal{C}_{S,p}^\mu &= \{\text{proc}_S(v) \mid v' \dashrightarrow v \text{ \& } v' \text{ is unmatched \& } \text{proc}_R(v') = p\} \cup \\
 &\quad \{\text{proc}_S(v) \mid v \text{ is unmatched \& } \text{proc}_R(v) = p\} \\
 \mathcal{C}_{R,p}^\mu &= \{\text{proc}_R(v) \mid v' \dashrightarrow v \text{ \& } v' \text{ is unmatched \& } \text{proc}_R(v') = p \text{ \& } v \text{ is matched}\}
 \end{aligned}$$

We can show that the $\text{ECG}(\mu)$ is acyclic if for all $p \in \mathbb{P}$, $p \notin \mathcal{C}_{R,p}^\mu$ (immediate consequence of Theorem 2 in [7]). Moreover, we write \mathbb{B}_{good} for the subset of \mathbb{B} formed by the tuples $(\mathcal{C}_{S,p}, \mathcal{C}_{R,p})_{p \in \mathbb{P}}$ such that $p \notin \mathcal{C}_{R,p}$ for all p .

Proposition 1 ([7]). For $w \in \Sigma^*$, $\text{msc}(w)$ satisfies causal delivery if and only if $\mathcal{B}(\mu(w)) \in \mathbb{B}_{\text{good}}$.

Noticing that \mathbb{B} is finite, we build an automaton $\mathcal{A}(B_0, B_1)$ with $B_0, B_1 \in \mathbb{B}$. The intuition behind these two buffer states is that B_0 summarises the conflict graph derived from previous exchanges and B_1 summarises the conflict graph obtained when a new exchange is added.

Definition 8 (Automaton of causal exchanges). *The automaton $\mathcal{A}(B_0, B_1)$ is defined as follows:*

- \mathbb{B} is the set of states,
- B_0 is the initial state (hereafter, we assume that $B_0 = (C_{S,p}^{(0)}, C_{R,p}^{(0)})_{p \in \mathbb{P}}$).
- $\{B_1\}$ is the set of final states
- the transition relation $(\xrightarrow{a})_{a \in \Sigma}$ is defined as follows:
 - $(C_{S,p}, C_{R,p})_{p \in \mathbb{P}} \xrightarrow{!v^{p \rightarrow q}} (C'_{S,p}, C'_{R,p})_{p \in \mathbb{P}}$ holds if for all $r \in \mathbb{P}$: let the intermediate set $C''_{S,r}$ be defined by

$$C''_{S,r} = \begin{cases} C_{S,r} \cup \{p\} & \text{if } p \in C_{R,r}^{(0)} \text{ or } q \in C_{R,r} \\ C_{S,r} & \text{otherwise} \end{cases}$$

Then

$$C'_{S,r} = \begin{cases} C''_{S,r} \cup C_{S,q} & \text{if } p \in C''_{S,r} \\ C_{S,r} & \text{otherwise} \end{cases} \quad \text{and} \quad C'_{R,r} = \begin{cases} C_{R,r} \cup \{q\} \cup C_{R,q} & \text{if } p \in C''_{S,r} \\ C_{R,r} & \text{otherwise} \end{cases}$$

- $(C_{S,p}, C_{R,p})_{p \in \mathbb{P}} \xrightarrow{!v^{p \rightarrow q}} (C'_{S,p}, C'_{R,p})_{p \in \mathbb{P}}$ holds if for all $r \in \mathbb{P}$,

$$C'_{S,r} = \begin{cases} C_{S,r} \cup \{p\} & \text{if } q = r \text{ or } q \in C_{R,r} \\ C_{S,r} & \text{otherwise} \end{cases} \quad \text{and} \quad C'_{R,r} = C_{R,r}$$

Let $\mathcal{L}(B_0, B_1)$ denote the language recognized by $\mathcal{A}(B_0, B_1)$.

Example 2. Consider $\mu_4 = msc(w)$ with $w = !v_3^{p_1 \rightarrow p_2} !v_4^{p_3 \rightarrow p_2} !v_5^{p_4 \rightarrow p_6} !v_6^{p_6 \rightarrow p_7}$ and assume we start with B_0 such that $C_{S,p_5} = \{p_4\}$ and $C_{R,p_5} = \{p_3\}$. Then the update of B (or, more precisely, of C_{S,p_5} , C_{R,p_5} , and C_{S,p_2}) after reading each message is shown below. Note how v_6 has no effect, despite the fact that $p_6 \in C_{R,p_5}$ at the time the message is read.

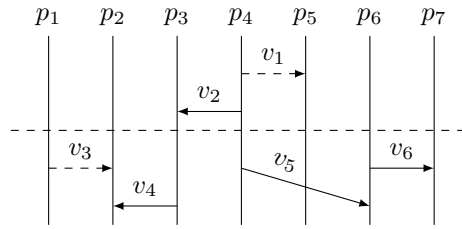


Fig. 7. MSC μ_4

$$\begin{array}{l} C_{S,p_5} \left| \begin{array}{ccccc} \{p_4\} & \{p_4\} & \{p_1, p_3, p_4\} & \{p_1, p_3, p_4\} & \{p_1, p_3, p_4\} \\ \{p_3\} & \xrightarrow{!v_3} \{p_3\} & \xrightarrow{!v_4} \{p_2, p_3\} & \xrightarrow{!v_5} \{p_2, p_3, p_6\} & \xrightarrow{!v_6} \{p_2, p_3, p_6\} \\ \emptyset & \{p_1\} & \{p_1\} & \{p_1\} & \{p_1\} \end{array} \right. \end{array}$$

Next lemma states that $\mathcal{A}(B, B')$ recognizes the words w such that $msc(w)$, starting with an initial buffer state B , ends in final buffer state B' .

Lemma 4. *Let $B, B' \in \mathbb{B}$ and $w \in \Sigma^*$. Then $w \in \mathcal{L}(B, B')$ if and only if for all MSC μ such that $B = \mathcal{B}(\mu)$, $B' = \mathcal{B}(\mu \cdot msc(w))$.*

Proof. Take $w = a_0 \dots a_n \in \Sigma^*$. To prove the lemma it is sufficient to show that if $B = \mathcal{B}(\mu)$ and :

$$B = (C_{S,p}^{(0)}, C_{R,p}^{(0)})_{p \in \mathbb{P}} \xrightarrow{a_0} (C_{S,p}^{(1)}, C_{R,p}^{(1)})_{p \in \mathbb{P}} \xrightarrow{a_1} \dots \xrightarrow{a_n} (C_{S,p}^{(n+1)}, C_{R,p}^{(n+1)})_{p \in \mathbb{P}} = B',$$

then $B' = \mathcal{B}(\mu \cdot msc(w))$.

The proof proceeds by induction on the length of w where the inductive hypothesis is that $(C_{S,p}^{(n)}, C_{R,p}^{(n)})_{p \in \mathbb{P}} = \mathcal{B}(\mu \cdot msc(a_0 \dots a_{n-1}))$.

We start by showing that $\forall r \in \mathbb{P}$, $C_{S,r}^{(n+1)} = \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$. Suppose that $p \in C_{S,r}^{(n+1)}$. If $p \in C_{S,r}^{(0)}$ then we can immediately conclude that $p \in \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$ since $C_{S,r}^{(0)} = \mathcal{C}_{S,r}^\mu \subseteq C_{S,r}^{(n+1)}$ and function $\mathcal{B}(\cdot)$ is increasing monotone. Instead if $p \notin C_{S,r}^{(0)}$ then the following can happen (without loss of generality suppose that p has been added while reading the last symbol of w):

$$- a_n = !?v^{p \rightarrow q} \text{ and } p \in C_{R,r}^{(0)} = \mathcal{C}_{R,r}^\mu$$

Then there exists a message in μ , v'' such that $\text{proc}_R(v'') = p$ and $v' \xrightarrow{SS} v''$ with v' unmatched. Then it is easy to see that $v'' \xrightarrow{SS} v$ and therefore $p \in \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$;

$$- a_n = !?v^{p \rightarrow q} \text{ and } q \in C_{R,r}^{(n)} = \mathcal{C}_{R,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$$

Then there exists a message v'' in $\mu \cdot msc(a_0 \dots a_{n-1})$, such that $\text{proc}_R(v'') = q$ and $v' \xrightarrow{SS} v''$ with v' unmatched. Then it is easy to see that $v'' \xrightarrow{SS} v$ and therefore $p \in \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$;

$$- a_n = !?v^{p' \rightarrow q} \text{ and } p' \in C_{S,r}^{(n)} \text{ and } p \in C_{S,q}^{(n)} = \mathcal{C}_{S,q}^{\mu \cdot msc(a_0 \dots a_{n-1})}$$

Then there exists a message v'' in $\mu \cdot msc(a_0 \dots a_{n-1})$, such that $\text{proc}_S(v'') = p$ and $v' \xrightarrow{SS} v''$ with v' unmatched and $\text{proc}_R(v') = q$. Then it is easy to see that $v \xrightarrow{SS} v''$ and since $p' \in C_{S,r}^{(n)}$ and with an analysis similar to the one above we have $v''' \xrightarrow{SS} v$ with v''' unmatched and we can conclude $p \in \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$.

$$- a_n = !?v^{p \rightarrow q} \text{ and } p \in C_{R,r}$$

Analogous to case 2 above.

$$- a_n = !?v^{p \rightarrow r}$$

We can immediately conclude $p \in \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$ as v is unmatched and $\text{proc}_R(v) = r$.

Now suppose that $p \in \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$ (without loss of generality we can assume $p \notin \mathcal{C}_{S,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$). Then either $a_n = !?v^{p \rightarrow r}$ then it is immediate to see that $p \in C_{S,r}^{(n+1)}$ or $a_n = !?v^{p \rightarrow q}$ and $v' \xrightarrow{SS} v$ for some v' unmatched and $\text{proc}_R(v') = r$. $p \notin \mathcal{C}_{S,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$ entails that either $q \in \mathcal{C}_{R,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$ or $p \in \mathcal{C}_{R,r}^\mu$ (notice

that since w is an exchange, $p \notin \mathcal{C}_{R,r}^{\mu \cdot msc(a_0 \dots a_{n-1})} \setminus \mathcal{C}_{R,r}^\mu$. In both cases we can conclude $p \in C_{S,r}^{(n+1)}$.

Next we show that $\forall r \in \mathbb{P}$, $C'_{R,r} = \mathcal{C}_{R,r}^{\mu \cdot msc(w)}$. Suppose that $p \in C_{R,r}^{(n+1)}$ (without loss of generality we can assume $p \notin C(n+1)_{R,r}$). This entails that:

- either $a_n = !?v^q \rightarrow p$ with $q \in C_{S,r}^{(n+1)} = \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$:
Then there exists a message v'' in $\mu \cdot msc(w)$, such that $\text{proc}_S(v'') = q$ and $v' \xrightarrow{SS} v''$ with v' unmatched and $\text{proc}_R(v') = r$. Then it is easy to see that $v'' \xrightarrow{SS} v$ and we can conclude $p \in \mathcal{C}_{R,r}^{\mu \cdot msc(w)}$;
- or $a_n = !?v^q \rightarrow p'$ with $q \in C_{S,r}^{(n+1)} = \mathcal{C}_{S,r}^{\mu \cdot msc(w)}$ and $p \in C_{R,p'}^{(n)} = \mathcal{C}_{R,p'}^{\mu \cdot msc(a_0 \dots a_{n-1})}$:
Then there exists a message v'' in $\mu \cdot msc(w)$, such that $\text{proc}_S(v'') = q$ and $v' \xrightarrow{SS} v''$ with v' unmatched and $\text{proc}_R(v') = r$. Similarly there is v''' in $\mu \cdot msc(a_0 \dots a_{n-1})$ such that $\text{proc}_R(v''') = p$ and $v^{iv} \xrightarrow{SS} v'''$ with v^{iv} unmatched and $\text{proc}_R(v^{iv}) = p'$. Now when adding v to the conflict graph we have $v \xrightarrow{SS} v^{iv}$. Hence we can conclude $p \in \mathcal{C}_{R,r}^{\mu \cdot msc(w)}$.

Now suppose that $p \in \mathcal{C}_{R,r}^{\mu \cdot msc(w)}$ (without loss of generality we can assume $p \notin \mathcal{C}_{R,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$). We know $\mathcal{C}_{R,r}^{\mu \cdot msc(a_0 \dots a_{n-1})} = C_{R,r}^{(n)}$ and let $a_n = !?v^q \rightarrow p$. The following can happen: $q \in \mathcal{C}_{S,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$, or $q \in \mathcal{C}_{R,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$, or $p \in \mathcal{C}_{S,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$. The last case is when $a_n = !?v^{q'} \rightarrow p'$ and $p \in \mathcal{C}_{R,p'}^{\mu \cdot msc(a_0 \dots a_{n-1})}$ and $q' \in \mathcal{C}_{S,r}^{\mu \cdot msc(a_0 \dots a_{n-1})}$. For all this cases, by inductive hypothesis and by Definition 8 we can conclude $p \in C_{R,r}^{n+1}$. \square

4.3 Language of reachable exchanges

The only thing that remains to do is to combine the previous automata to define one that recognizes the (codings of) reachable exchanges. The language $\mathcal{L}(\text{SR}(\mathbf{in}, \mathbf{mid}, \mathbf{fin}))$ contains arbitrary exchanges which do not necessarily satisfy causal delivery. Here comes into play the $\mathcal{A}(B, B')$ automata, where we take B and $B' \in \mathbb{B}_{good}$ in order to ensure causal delivery.

Let

$$\mathcal{L}_c(\mathbf{in}, \mathbf{fin}, B, B') \stackrel{\text{def}}{=} \bigcup_{\mathbf{mid} \in L_\otimes} \mathcal{L}(\text{SR}(\mathbf{in}, \mathbf{mid}, \mathbf{fin})) \cap \mathcal{L}(B, B').$$

Intuitively, $\mathcal{L}_c(\mathbf{in}, \mathbf{fin}, B, B')$ is the language of (codings of) exchanges between global states \mathbf{in} and \mathbf{fin} starting with an initial buffer state B and ending in final buffer state B' ; when moreover $B, B' \in \mathbb{B}_{good}$, these exchanges satisfy causal delivery.

The last step is to combine causal delivery exchanges so that they can be performed by the system one after the other from the initial state \mathbf{l}_0 . This motivates the definition of the following set \mathcal{R} of *reachable languages*. Let $B_\emptyset = (\emptyset, \emptyset)_{p \in \mathbb{P}}$.

Definition 9 (Reachable languages). Given a system $\mathfrak{S} = (L_{\mathfrak{S}}, \delta_{\mathfrak{S}}, \mathbf{l}_0)$, the set \mathcal{R} of reachable languages is the least set of languages of the form $\mathcal{L}_c(\mathbf{in}, \mathbf{fin}, B_i, B_f)$ defined as follows.

1. for any $\mathbf{l} \in L_{\mathfrak{S}}$ and $B \in \mathbb{B}_{good}$, $\mathcal{L}_c(\mathbf{l}_0, \mathbf{l}, B_{\emptyset}, B)$ is in \mathcal{R}
2. for any $\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3 \in L_{\mathfrak{S}}$ and any $B_1, B_2, B_3 \in \mathbb{B}_{good}$, if $\mathcal{L}_c(\mathbf{l}_1, \mathbf{l}_2, B_1, B_2) \in \mathcal{R}$ and $\mathcal{L}_c(\mathbf{l}_1, \mathbf{l}_2, B_1, B_2) \neq \emptyset$ then $\mathcal{L}_c(\mathbf{l}_2, \mathbf{l}_3, B_2, B_3) \in \mathcal{R}$.

Then the union $\bigcup \mathcal{R}$ of all reachable languages is equal to the language $\mathcal{L}_r = \{w \in \Sigma^* \mid msc(w) \text{ is reachable}\}$. As a consequence, we get the following result.

Theorem 2. \mathcal{L}_r is a regular language and is accepted by an effective finite state automaton.

Proof. $\Rightarrow w \in \bigcup \mathcal{R}$ so there is a sequence of words $w_1, \dots, w_n \in \Sigma^*$ such that $\forall 1 \leq j \leq n, w_j \in \mathcal{L}_c(\mathbf{l}_{j-1}, \mathbf{l}_j, B_{j-1}, B_j) \neq \emptyset$ with $B_j \in \mathbb{B}_{good}$, and there is also $\mathbf{fin} \in L_{\mathfrak{S}}, B_f \in \mathbb{B}_{good}$ such that $w \in \mathcal{L}_c(\mathbf{l}_n, \mathbf{fin}, B_n, B_f)$.

So, $w_j \in \mathcal{L}(\mathbf{SR}(\mathbf{l}_{j-1}, \mathbf{l}'_j, \mathbf{l}_j))$ for $\mathbf{l}'_j \in L_{\mathfrak{S}}$, and, by Lemma 3, $\mathbf{l}_{j-1} \xrightarrow{msc(w_j)} \mathbf{l}_j$, and $\mathbf{l}_n \xrightarrow{msc(w)} \mathbf{fin}$.

Moreover, for all $1 \leq j \leq n$, $w_j \in \mathcal{L}(B_{j-1}, B_j)$ and $B_j \in \mathbb{B}_{good}$. So, each $msc(w_j)$ verifies causal delivery, and, we can easily show by induction that $msc(w_1) \dots msc(w_n) \cdot msc(w)$ verifies causal delivery too. Finally, by Proposition ??, $msc(w_1) \dots msc(w_n) \cdot msc(w) \in asTr(\mathfrak{S})$ and so, $msc(w)$ is reachable.

$\Leftarrow msc(w)$ is reachable so there is $\mu_1 \dots \mu_n$ a sequence of MSCs such that $\mu_1 \dots \mu_n \cdot msc(w) \in asTr(\mathfrak{S})$. Suppose that $\mu_1 \dots \mu_n = \varepsilon$, then, $\mathbf{l}_0 \xrightarrow{msc(w)} \mathbf{fin}$ for some \mathbf{fin} . Therefore, $w \in \mathcal{L}(\mathbf{SR}(\mathbf{l}_0, \mathbf{l}, \mathbf{fin}))$ for some $\mathbf{l} \in L_{\mathfrak{S}}$. As $msc(w) \in asTr(\mathfrak{S})$, $msc(w)$ verifies causal delivery and so there is B such that $w \in \mathcal{L}(B_0, B')$ with $B' \in \mathbb{B}_{good}$. Finally, we have that $w \in \mathcal{L}_c(\mathbf{l}_0, \mathbf{fin}, B_0, B)$ and so $w \in \bigcup \mathcal{R}$.

Now, suppose that $\mu_1 \dots \mu_n \neq \varepsilon$. Then, there is a sequence $w_1, \dots, w_n \in \Sigma^*$ such that $msc(w_i) = \mu_i$, $1 \leq i \leq n$, and we can suppose that $w_1, \dots, w_n \in \mathcal{L}_r$. By Lemma 4, there is $B = \mathcal{B}(\mu_1 \dots \mu_n), B' = \mathcal{B}(\mu_1 \dots \mu_n \cdot msc(w)) \in \mathbb{B}_{good}$ such that $w \in \mathcal{L}(B, B')$. Moreover, there is $\mathbf{in}, \mathbf{fin}$ such that $\mathbf{in} \xrightarrow{msc(w)} \mathbf{fin}$, so $w \in \mathcal{L}(\mathbf{SR}(\mathbf{in}, \mathbf{mid}, \mathbf{fin}))$ for some \mathbf{mid} . Finally, we have $w \in \mathcal{L}_c(\mathbf{in}, \mathbf{fin}, B, B')$ and then $w \in \mathcal{L}_r$. \square

5 Prime exchanges

We reformulate the primality of an exchange in terms of its conflict graph. We say that the conflict graph $CG(\mu)$ associated with the MSC μ is strongly connected if for all $v, v' \in V$ it holds that $v \rightarrow^* v'$, where \rightarrow^* is the reflexive transitive closure of $\rightarrow = \bigcup_{X, Y \in \{S, R\}} \xrightarrow{XY}$.

Lemma 5. *An exchange μ is prime iff $\text{CG}(\mu)$ is strongly connected.*

Proof. Let $\mu = \mu_1 \cdots \mu_n$ be an MSC formed with a sequence of exchanges. Let e, e' be two events of μ , and let $i, i' \in \{1, \dots, n\}$ be such that e appears in μ_i and e' appears in $\mu_{i'}$. If there is an edge $e \xrightarrow{XY} e'$ in the conflict graph of μ , then $i \leq i'$. As a consequence, if e and e' are on a same strongly connected component, then $i = i'$, and if the conflict graph of μ is strongly connected, then $n = 1$ and μ is a prime exchange. \square

Next we discuss the construction of the automaton that recognizes $\{w \in \Sigma^* \mid \text{msc}(w) \text{ is prime}\}$. Since there are infinitely many $\text{CG}(\text{msc}(w))$, in order to have a finite state automaton, we compute a finite abstractions of $\text{CG}(\text{msc}(w))$ that is sound in the sense that $\text{CG}(\text{msc}(w))$ is strongly connected if and only if its abstraction is of a certain shape. Let us now define this abstraction.

We need to define some graph transformations. The graphs we are going to manipulate are oriented graphs labeled with a pair of set of processes on each vertex. We call such objects P-graphs. Formally, a P-graph is a tuple $(V, E, \lambda_S, \lambda_R)$ with $E \subseteq V \times V$ and $\lambda_X : V \rightarrow 2^P$ for $X \in \{S, R\}$. The P-graph $\text{pgr}(\mu)$ associated with the conflict graph $\text{CG}(\mu) = (V, \{\xrightarrow{XY}\}_{X,Y \in \{S,R\}})$ is $(V, E, \lambda_S, \lambda_R)$ where (1) $(v, v') \in E$ if $v \xrightarrow{XY} v'$ for some X, Y , (2) $\lambda_S(v) = \{\text{proc}_S(v)\}$, and (3) if v is matched, then $\lambda_R(v) = \{\text{proc}_R(v)\}$, and if v is unmatched $\lambda_R(v) = \emptyset$.

The first graph transformation we consider consists in merging the vertices that belong to a same strongly connected component (SCC). Formally, let $G = (V, E, \lambda_S, \lambda_R)$ be a P-graph, and let $\text{merge}(G) = (V', E', \lambda_S, \lambda_R)$ be defined by (1) V' is the set of maximal SCCs of G , (2) for two distinct maximal SCCs U, U' , $(U, U') \in E'$ if there are $v \in U$ and $v' \in U'$ such that $(v, v') \in E^+$ (the transitive closure of E), (3) for $X = S, R$, $\lambda_X(U) = \bigcup_{v \in U} \lambda_X(v)$.

The second graph transformation we consider consists in erasing some of the processes that appear in the labels. Let $G = (V, E, \lambda_S, \lambda_R)$ be a fixed P-graph, and let $v \in V$, $X \in \{S, R\}$, and $p \in \lambda_X(v)$ be fixed. We say that p is X-redundant in v if there are v_1, v_2 such that (1) $(v_1, v) \in E^+$ and $(v, v_2) \in E^+$, and (2) $p \in \lambda_X(v_1) \cap \lambda_X(v_2)$. Intuitively, p is redundant in v if it also appears in the label of an ancestor and a descendant of v . We define the P-graph $\text{erase}(G)$ as $(V, E, \lambda'_S, \lambda'_R)$ where for all $X \in \{S, R\}$, for all $v \in V$, $\lambda'_X(v)$ is the set of processes $p \in \lambda_X(v)$ such that p is not X-redundant at v .

The last graph transformation we consider consists in sweeping out the vertices labeled with empty sets of processes. Formally, for $G = (V, E, \lambda_S, \lambda_R)$, the P-graph $\text{sweep}(G)$ is $(V', E', \lambda_S, \lambda_R)$ where $V' = \{v \in V \mid \lambda_S(v) \cup \lambda_R(v) \neq \emptyset\}$ and $E' = E \cap V' \times V'$. The abstraction $\alpha(G)$ of a P-graph G is defined as $\text{sweep}(\text{erase}(\text{merge}(G)))$. An example of the construction is in Fig 8.

Lemma 6. *$\text{CG}(\mu)$ is strongly connected iff $\alpha(\text{pgr}(\mu))$ is a single vertex graph.*

Proof. By definition of α , and in particular of function $\text{merge}(\cdot)$, a vertex of $\alpha(\text{pgr}(\mu))$ corresponds to a strongly connected component of $\text{CG}(\mu)$. \square

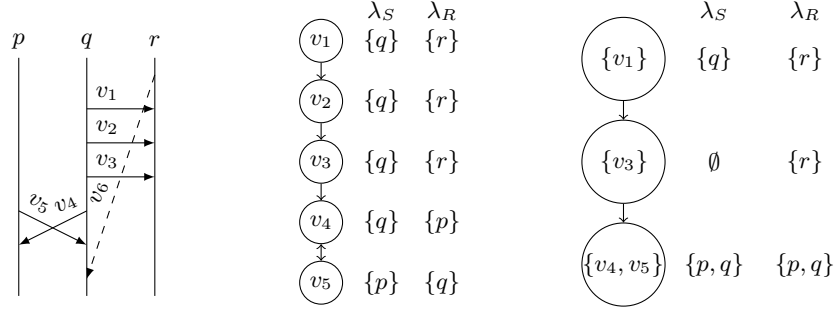


Fig. 8. MSC μ_5 , its associated P-graph $\text{pgr}(\mu_5)$, and the abstraction $\alpha(\text{pgr}(\mu_5))$.

By construction, for any process p , and for any $X \in \{S, R\}$, there are at most two vertices v of $\alpha(\text{pgr}(\mu))$ such that $p \in \lambda_X(v)$. From this, we deduce that $\alpha(\text{pgr}(\mu))$ has at most $2|\mathbb{P}|$ vertices, and as a consequence:

Lemma 7. $\#\{\alpha(\text{pgr}(\mu)) \mid \mu \text{ is an exchange}\} \leq 2^{6|\mathbb{P}|^2}$.

Proof. Let $n \geq 0$ be fixed and let us give an upper bound on the number of P-graphs with n vertices. First, there are $2^{n(n-1)}$ different possible choices for the edge relation. By construction (in particular, by definition of function $\text{erase}(\cdot)$), it holds that

$$(1) \quad \forall p \in \mathbb{P}, \forall X \in \{S, R\}, \#\{v \mid p \in \lambda_X(v)\} \leq 2.$$

A choice for the λ function is therefore the choice, for each p , of at most two vertices v such that $p \in \lambda_S(v)$ and at most two other vertices v such that $p \in \lambda_R(v)$. So there are at most n^4 different choices for each p , and at most $n^{4|\mathbb{P}|}$ different choices for λ . To sum up, there are less than $2^{n^2} n^{4|\mathbb{P}|}$ P-graphs with n vertices.

Now, again from (1), there are at most $2|\mathbb{P}|$ vertices in a P-graph, so the number of P-graph is bounded by

$$\sum_{n=1}^{2|\mathbb{P}|} 2^{n^2} n^{4|\mathbb{P}|} \leq 2|\mathbb{P}| 2^{|\mathbb{P}|^2} (2|\mathbb{P}|)^{4|\mathbb{P}|} \leq 2^{|\mathbb{P}|^2} 2^{|\mathbb{P}|^2} (2^{|\mathbb{P}|})^{4|\mathbb{P}|} = 2^{6|\mathbb{P}|^2}.$$

□

There are therefore finitely many $\alpha(\text{pgr}(\mu))$. This allows us to define the automaton that computes $\alpha(\text{pgr}(\text{msc}(w)))$ for any $w \in \Sigma^*$ and accepts w in the language of this new automaton when this P-graph is a single vertex graph. Let $G = (V, E, \lambda_S, \lambda_R)$ and a letter $\dagger v^{p \rightarrow q} \in \Sigma$ be fixed. We want to define the transition function δ_g of our automaton, or in other words, the P-graph $\delta_g(G, \dagger v^{p \rightarrow q})$

reached after adding the message $\dagger v^{p \rightarrow q}$ to the MSC. We let $\delta_g(G, \dagger v_0^{p \rightarrow q}) = \alpha(G')$. $G' = (V', E', \lambda'_S, \lambda'_R)$ is defined as follows: (1) $V' = V \uplus \{v_0\}$, (2) $\lambda'_S(v_0) = \{p\}$, (3) if $\dagger = !?$, then $\lambda'_R(v_0) = \{q\}$, and if $\dagger = !$, then $\lambda'_R(v_0) = \emptyset$, (4) for all $v \in V$, for all $X \in \{S, R\}$, $\lambda'_X(v) = \lambda_X(v)$, and (5) the set of edges E' is defined as

$$E' = E \cup \{(v, v_0) \mid p \in \lambda_S(v)\} \cup \{(v_0, v) \mid p \in \lambda_R(v)\} \\ \cup \begin{cases} \{(v, v_0) \mid q \in \lambda_S(v) \cup \lambda_R(v)\} & \text{if } \dagger = !? \\ \emptyset & \text{if } \dagger = ! \end{cases}$$

For example, consider the MSC μ of Fig. 8 and let $G = \alpha(\text{pgr}(\mu))$ be its associated abstracted P-graph. Let G' be defined as above while reading $!v_6^{r \rightarrow q}$. Then G' is the graph on the right, and $\delta_g(G, v_6^{r \rightarrow q})$ is a single vertex graph.

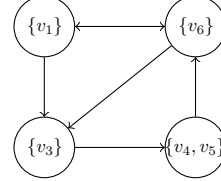


Fig. 9. Graph G'

Lemma 8. $\delta_g(\alpha(\text{pgr}(\text{msc}(w))), \dagger v^{p \rightarrow q}) = \alpha(\text{pgr}(\text{msc}(w \cdot \dagger v^{p \rightarrow q})))$.

Before we prove Lemma 8, we need to introduce a few notions and observations. Let $G = (V, E, \lambda_S, \lambda_R)$ be a P-graph. A vertex $v \in V$ is X -covered if for all $p \in \lambda_X(v)$ p is X -redundant. We also say that $v \in V$ is covered if it is both S -covered and R -covered. A partial abstraction of G is a graph $G' = (V', E', \lambda_S, \lambda_R)$ such that

- $V' = \{V_1, \dots, V_n\}$ where each V_i is a (not necessarily maximal) strongly connected component of G , all V_i are disjoint, and for all $v \in V \setminus \bigcup_{i=1}^n V_i$ v is covered.
- for all i, j , $(V_i, V_j) \in E'$ iff $(v, v') \in E$ for some $v \in V_i$ and some $v' \in V_j$.
- for all i, X , $\lambda_X(V_i) = \bigcup_{v \in V_i} \lambda_X(v)$

Intuitively, G' is a partial abstraction of G if it results from a “partial application” of the functions $\text{merge}(\cdot)$, $\text{erase}(\cdot)$, and $\text{sweep}(\cdot)$: some vertices of a same SCC are merged, but not necessarily all, some labels are erased, but not necessarily all, and some vertices are swepted, but not necessarily all. From this observation, it follows the following: if G' is a partial abstraction of G , then $\alpha(G') = \alpha(G)$.

Proof. Let $w \in \Sigma^*$ and $\dagger v^{p \rightarrow q}$ be fixed. Let $G_1 = \text{pgr}(\text{msc}(w))$ and $G_2 = \text{pgr}(\text{msc}(w \cdot \dagger v^{p \rightarrow q}))$ and let us compare G_1 and G_2 . First, there is an extra vertex v_0 in G_2 that represents $\dagger v^{p \rightarrow q}$, with $\lambda_S(v_0) = \{p\}$ and either $\lambda_R(v_0) = \{q\}$ (if $\dagger = !?$) or $\lambda_R(v_0) = \emptyset$ (if $\dagger = !$). Now, consider the extra edges. Obviously, these extra edges have v_0 either as source or as destination. First consider the edges with v_0 as destination. The send event of v_0 happens after all send events of p , so for all $v \neq v_0$ such that $p \in \lambda_S(v)$, $(v, v_0) \in E_2$. In the case where $\dagger = !?$, the receive event of v_0 also happens after all send and receive events of q , so for all $v \neq v_0$ such that $q \in \lambda_S(v) \cup \lambda_R(v)$, $(v, v_0) \in E_2$. There are no other incoming edges in v_0 . Now, consider the outgoing edges of v_0 . The send event of v_0 happens before all receive events of p , so for all $v \neq v_0$ such that $p \in \lambda_R(v)$, $(v_0, v) \in E_2$. To sum up, we have:

$$\begin{aligned}
 E_2 = & E_1 \cup \{(v, v_0) \mid p \in \lambda_S(v)\} \\
 & \cup \{(v_0, v) \mid p \in \lambda_R(v)\} \\
 & \cup \begin{cases} \{(v, v_0) \mid q \in \lambda_S(v) \cup \lambda_R(v)\} & \text{if } \dagger = !? \\ \emptyset & \text{if } \dagger = ! \end{cases}
 \end{aligned}$$

Observe now that the rules to add vertices and edges to go from G_1 to G_2 are exactly the same as the rules to go from G to G' in the definition of $\delta_g(G, \dagger v^{p \rightarrow q})$. Assume that $G = \alpha(G_1) = \alpha(\text{pgr}(\text{msc}(w)))$. Then G' is a partial abstraction of G_2 . So by the discussion above,

$$\alpha(G') = \alpha(G_2).$$

Now, by definition of δ_g , $\delta_g(G, \dagger v^{p \rightarrow q}) = \alpha(G')$. To sum up

$$\delta_g(G, \dagger v^{p \rightarrow q}) = \alpha(G_2) = \alpha(\text{pgr}(\text{msc}(w \cdot \dagger v^{p \rightarrow q}))).$$

□

Theorem 3. *There is an effective deterministic finite state automaton \mathcal{A} with less than $2^{6|\mathbb{P}|^2}$ states such that $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \text{msc}(w) \text{ is prime}\}$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta_q, q_0, F)$ be defined by

- $Q = \{\alpha(\text{pgr}(\mu)) \mid \mu \text{ is an exchange}\};$
- δ_q as defined in Section 5;
- $q_0 = \alpha(\text{pgr}(\epsilon))$ where ϵ denotes the empty MSC
- $F = \{G \in Q \mid |G| = 1\}$

Then by Lemma 7, \mathcal{A} is a deterministic finite state automaton with at most $2^{6|\mathbb{P}|^2}$ states. Moreover, by Lemma 8, for all w , $\delta_q^*(q_0, w) = \text{pgr}(\text{msc}(w))$, so w is accepted iff $\text{pgr}(\text{msc}(w))$ is a single vertex graph. By Lemma 6, this is equivalent to the fact that $\text{msc}(w)$ is prime. □

6 Computation of k

So far we have shown: in Lemma 1, we established that a way to compute $\text{sd}(\mathfrak{S})$ was to compute the length k of the largest prime reachable exchange. To every word $w \in \Sigma^*$, we associated an MSC $\text{msc}(w)$, and we showed that for every reachable MSC μ , there exists a word $w \in \Sigma^*$ such that $\mu = \text{msc}(w)$ (Lemma 2). We deduced that k corresponds to the length of the longest word of $\mathcal{L}_\mathbf{r} \cap \mathcal{L}_\mathbf{p}$, if $\mathcal{L}_\mathbf{r} \cap \mathcal{L}_\mathbf{p}$ is finite, otherwise $k = \infty$. In Section 4, we showed that $\mathcal{L}_\mathbf{r}$ is an effective regular language, and, in Section 5, we showed that $\mathcal{L}_\mathbf{p}$ is also an effective regular language. We deduce that $\mathcal{L}_\mathbf{r} \cap \mathcal{L}_\mathbf{p}$ is therefore an effective regular language, and that k is computable (since the finiteness and the length of the longest word of a regular language are computable). With a careful analysis of the automata that come into play, we can give an upper bound on k .

Theorem 4. *$\text{sd}(\mathfrak{S})$ is computable, and if $\text{sd}(\mathfrak{S}) < \infty$ then $\text{sd}(\mathfrak{S}) < |\mathfrak{S}|^{2^{8|\mathbb{P}|^2}}$, where $|\mathfrak{S}|$ is the number of global control states and $|\mathbb{P}|$ the number of processes.*

Proof. The fact that $\text{sd}(\mathfrak{S})$ is computable is explained at the beginning of Section 6. We therefore only prove the claim that, when $k < \infty$ it holds that $k < |\mathfrak{S}|^{2^{8|\mathbb{P}|^2}}$. k is the length of the longest word in $\mathcal{L}_r \cap \mathcal{L}_p$. By Theorem 2,

$$\mathcal{L}_r = \bigcup \mathcal{R}$$

and by Theorem 3 there is an automaton \mathcal{A} such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}_p$. So we need to bound the length of the longest word of

$$\mathcal{L}(\text{SR}(\mathbf{l}, \mathbf{mid}, \mathbf{l}')) \cap \mathcal{L}(\mathcal{A}(B, B')) \cap \mathcal{L}(\mathcal{A})$$

assuming that this language is finite, for any $\mathbf{l}, \mathbf{mid}, \mathbf{l}', B, B'$. This bound is given by the number of states of any automaton that accepts this language (since any longer word would require the automaton to feature a loop, and the language would not be finite). This language is recognized by an automaton that is the product of the automata $\text{SR}(\mathbf{l}, \mathbf{mid}, \mathbf{fin})$, $\mathcal{A}(B, B')$, and \mathcal{A} , so its number of states is bounded by

$$|\text{SR}(\mathbf{l}, \mathbf{mid}, \mathbf{l}')| \times |\mathcal{A}(B, B')| \times |\mathcal{A}|$$

By definition of SR , $L_{\text{SR}} = L_{\mathfrak{S}}^2$, so $|\text{SR}| \leq |L_{\mathfrak{S}}|^2$ (which we can also write $|\mathfrak{S}|^2$). By definition of $\mathcal{A}(B, B')$, $L_{(B, B')} = \mathbb{B} = (2^{\mathbb{P}} \times 2^{\mathbb{P}})^{\mathbb{P}}$, so $|\mathcal{A}(B, B')| \leq 2^{2|\mathbb{P}|^2}$. Finally, by Theorem 3, $|\mathcal{A}| \leq 2^{6|\mathbb{P}|^2}$. All together,

$$\begin{aligned} k &\leq |\mathfrak{S}|^2 2^{2|\mathbb{P}|^2} 2^{6|\mathbb{P}|^2} \\ &\leq |\mathfrak{S}|^{2^{8|\mathbb{P}|^2}} \end{aligned}$$

□

As an immediate consequence of Theorems 1 and 4, we get the following.

Theorem 5. *The following problem is decidable : given a system \mathfrak{S} , does there exists a k such that \mathfrak{S} is k -synchronizable.*

7 Conclusion

We established that it is possible to determine whether there exists a bound k such that a given communicating system is k -synchronizable. For this, we showed how the set of sequences of actions that compose an exchange of arbitrary size can be represented as a regular language, which was possible thanks to the mailbox semantics of communications. We leave for future work to decide whether it would be possible to extend our result to peer-to-peer semantics.

References

1. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. *Inf. Comput.* **127**(2), 91–101 (1996)
2. Basu, S., Bultan, T.: On deciding synchronizability for asynchronously communicating systems. *Theor. Comput. Sci.* **656**, 60–75 (2016)
3. Bouajjani, A., Enea, C., Ji, K., Qadeer, S.: On the completeness of verifying message passing programs under bounded asynchrony. In: *CAV 2018*. pp. 372–391 (2018)
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. *J. ACM* **30**(2), 323–342 (1983)
5. Chaouch-Saad, M., Charron-Bost, B., Merz, S.: A reduction theorem for the verification of round-based distributed algorithms. In: *RP 2009*. vol. 5797, pp. 93–106. Springer (2009)
6. Chou, C., Gafni, E.: Understanding and verifying distributed algorithms using stratified decomposition. In: *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, 1988. pp. 44–65. ACM (1988)
7. Di Giusto, C., Laversa, L., Lozes, É.: On the k-synchronizability of systems. In: Goubault-Larrecq, J., König, B. (eds.) *FOSSACS 2020, ETAPS 2020*. vol. 12077, pp. 157–176. Springer (2020)
8. Elrad, T., Francez, N.: Decomposition of distributed programs into communication-closed layers. *Sci. Comput. Program.* **2**(3), 155–173 (1982)
9. Finkel, A., Lozes, É.: Synchronizability of communicating finite state machines is not decidable. In: *ICALP 2017*. pp. 122:1–122:14 (2017)
10. Genest, B., Kuske, D., Muscholl, A.: On communicating automata with bounded channels. *Fundam. Inform.* **80**(1-3), 147–167 (2007)
11. von Gleissenthall, K., Kici, R.G., Bakst, A., Stefan, D., Jhala, R.: Pretend synchrony: synchronous verification of asynchronous distributed programs. *PACMPL* **3**(POPL), 59:1–59:30 (2019)
12. Kragl, B., Qadeer, S., Henzinger, T.A.: Synchronizing the asynchronous. In: *CONCUR 2018*. vol. 118, pp. 21:1–21:17 (2018)
13. La Torre, S., Madhusudan, P., Parlato, G.: Context-bounded analysis of concurrent queue systems. In: *TACAS 2008, ETAPS 2008*. pp. 299–314 (2008)
14. Lipton, R.J.: Reduction: A method of proving properties of parallel programs. *Commun. ACM* **18**(12), 717–721 (1975)