On the k-synchronizability for mailbox systems

Cinzia Di Giusto Laetitia Laversa
Etienne Lozes
Université Côte d'Azur, CNRS, I3S, France

September 5, 2019

Abstract

Asynchronous bounded or unbounded message passing is ubiquitous in communication-centric systems. When modelling distributed scenarios, it is important to understand whether buffers are bounded or not. In this paper, we work on the notion of k-synchronizability: a system is k-synchronizable if any of its executions, up to reordering causally independent actions, can be divided into a succession of k-bounded interaction phases. We show two results: first, the reachability problem is decidable for k-synchronizable systems; second, the membership problem (whether a given system is k-synchronizable) is decidable as well. Our proofs fix several important issues in previous attempts to prove these two results.

1 Introduction

Asynchronous message-passing is ubiquitous in communication-centric systems; these include high-performance computing, distributed memory management, event-driven programming, or web services orchestration. One of the parameters that plays an important role in these systems is whether the number of pending sent messages can be bounded in a predictable fashion, or whether the buffering capacity offered by the communication layer should be unlimited. Clearly, when considering implementation, testing, or verification, bounded asynchrony is preferred to unbounded asynchrony. Indeed, for bounded systems, reachability analysis and invariants inference can be solved by regular model-checking [5]. On the other hand, especially when designing a new system, it is easier to assume that the buffering capacity is unbounded, or that the bound is not known a priori. Thus, a question that arises naturally is whether a given system with k-bounded buffers has the same "behaviour" as the same system with unbounded buffers.

In a recent work [4], Bouajjani *et al.* introduced the notion of k-synchronizable system of finite state machines communicating through mailboxes. Intuitively, a system is k-synchronizable if any of its executions, up to reordering causally independent actions, can be chopped into a succession of k-bounded interaction phases. Each of these phases starts with at most k send actions that are followed by at most k receptions. The main motivation for k-synchronizable system is that the reachability problem is decidable.

As explained in the present paper, this result, although valid, is surprisingly non-trivial, mostly due to complications introduced by the mailbox semantics of communications. Some of these complications were missed by Bouajjani *et al.* and the algorithm for the reachability problem in [4] suffers from false positives (see discussion in Section 6). Another problem is the membership problem for the subclass of k-synchronizable systems: for a given k and a given system of communicating finite state machines, is this system k-synchronizable? The main result in [4] is that this problem is decidable. However, again, the proof of this result contains an important flaw at the very first step that breaks all subsequent developments; as a consequence, the algorithm given in [4] produces both false positives and false negatives.

In this work, we present a new proof of the decidability of the reachability problem together with a new proof of the decidability of the membership problem. Quite surprisingly, the reachability problem is more demanding in terms of causality analysis, whereas the membership problem, although rather intricate, builds on a simpler dependency analysis.

Outline. The next section recalls the definition of communicating systems and related notions. In Section 3 we introduce k-synchronizability and we give a graphical characterisation of this property. This characterisation corrects Theorem 1 in [4] and highlights the flaw in the proof of the membership problem. Next, in Section 4, we establish the decidability of the reachability problem, which is the core of our contribution, and in Section 5, we show the decidability of the membership problem. Section 6 discusses how our work is related to [4] and finally Section 7 concludes the paper discussing other related works.

2 Preliminaries

A mailbox communicating automaton is a finite state machine where transitions are labelled with either send or receive actions. Such an automaton may receive messages from other automata. Messages await to be received in a mailbox: a FIFO queue that stores all messages sent to a same automaton, regardless of their senders.

Let $\mathbb{V}$ be a finite set of messages and $\mathbb{P}$ a finite set of processes. A send action, denoted $send(p, q, \mathbf{v})$, designates the sending of message $\mathbf{v}$ from process p to process q. Similarly a receive action $rec(p, q, \mathbf{v})$ expresses that process q is receiving message $\mathbf{v}$ from p. We write a to denote a send or receive action. Let $S = \{send(p, q, \mathbf{v}) \mid p, q \in \mathbb{P}, \mathbf{v} \in \mathbb{V}\}$ be the set of send actions and $R = \{rec(p, q, \mathbf{v}) \mid p, q \in \mathbb{P}, \mathbf{v} \in \mathbb{V}\}$ 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 the parallel composition of processes.

Definition 1 (System). *A system is a tuple $\mathfrak{S} = \big((L_p, \delta_p, l_p^0) \mid p \in \mathbb{P}\big)$ 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 function (also denoted $l \xrightarrow{a}_p l'$) and l_p^0 is the initial state.*

Definition 2 (Configuration). *Let $\mathfrak{S} = \big((L_p, \delta_p, l_p^0) \mid p \in \mathbb{P}\big)$, a configuration is a pair $(\vec{l}, \mathtt{Buf})$ where $\vec{l} = (l_p)_{p \in \mathbb{P}} \in \Pi_{p \in \mathbb{P}} L_p$ is a global control state of $\mathfrak{S}$ (a local control state for each automaton), and $\mathtt{Buf} = (b_p)_{p \in \mathbb{P}} \in (\mathbb{V}^*)^{\mathbb{P}}$ is a vector of buffers, each b_p being a word over $\mathbb{V}$.*

We write $\vec{l_0}$ to denote the vector of initial states of all processes $p \in \mathbb{P}$, and

$\mathtt{Buf}_0$ stands for the vector of empty buffers. The semantics of a system is defined by the two rules below.

$$\frac{l_p \xrightarrow{send(p,q,\mathbf{v})}_p l'_p \quad b'_q = b_q \cdot \mathbf{v}}{(\vec{l}, \mathtt{Buf}) \xrightarrow{send(p,q,\mathbf{v})} (\vec{l}[l'_p/l_p], \mathtt{Buf}[b'_q/b_q])} \qquad \frac{l_q \xrightarrow{rec(p,q,\mathbf{v})}_q l'_q \quad b_q = \mathbf{v} \cdot b'_q}{(\vec{l}, \mathtt{Buf}) \xrightarrow{rec(p,q,\mathbf{v})} (\vec{l}[l'_q/l_q], \mathtt{Buf}[b'_q/b_q])}$$

A send action adds a message in the buffer b of the receiver, and a receive action pops the message from this buffer. An execution $e = a_1 \dots a_n$ is a sequence of actions in $S \cup R$ such that $(\vec{l_0}, \mathtt{Buf}_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (\vec{l}, \mathtt{Buf})$ for some $\mathtt{Buf}$. As usual $\overset{e}{\Rightarrow}$ stands for $\xrightarrow{a_1} \dots \xrightarrow{a_n}$. We write $asEx(\mathfrak{S})$ to denote the set of executions of a system $\mathfrak{S}$. In a sequence of actions $e = a_1 \cdots a_n$, a send action $a_i = send(p, q, \mathbf{v})$ is *matched* by a reception $a_j = rec(p', q', \mathbf{v}')$ (denoted by $a_i \vdash\!\dashv a_j$) if $p = p'$, $q = q'$, $\mathbf{v} = \mathbf{v}'$, and there is $\ell \geq 1$ such that a_i and a_j are the ℓth actions of e with these properties respectively. A send action a_i is *unmatched* if there is no matching reception in e. A *message exchange* of a sequence of actions e is a set either of the form $v = \{a_i, a_j\}$ with $a_i \vdash\!\dashv a_j$ or of the form $v = \{a_i\}$ with a_i unmatched. When v is either an unmatched $send(p, q, \mathbf{v})$ or a pair of matched actions $\{send(p, q, \mathbf{v}), rec(p, q, \mathbf{v})\}$, we write $\mathsf{proc}_S(v)$ for p and $\mathsf{proc}_R(v)$ for q. Note that $\mathsf{proc}_R(v)$ is defined even if v is unmatched. Finally, we write $\mathsf{procs}(v)$ for $\{p\}$ in the case of an unmatched send and $\{p, q\}$ in the case of a matched send.

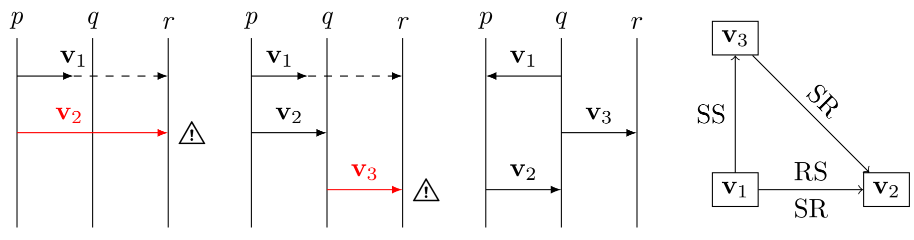
An execution imposes a total order on the actions. On the other hand, a message sequence chart (MSC) will only impose an order between matched pairs of actions and between the actions of a same process. Informally, a MSC will be depicted with vertical timelines (one for each process) that carry some points representing send and receive events of this process. An arc is drawn between two matched events. We will also draw a dashed arc to depict an unmatched send event. A MSC is a partially ordered set of events, each corresponding to a send or receive action. For a given sequence of actions $e = a_1 \dots a_n$, we let po be the set of pairs of indices $(i, j) \in [1..n]^2$ such that $i < j$ and a_i and a_j are actions of a same process, i.e., there is $p \in \mathbb{P}$ such that $\{a_i, a_j\} \subseteq S_p \cup R_p$. We also write src for the set of pairs of indices (i, j) such that $a_i \vdash\!\dashv a_j$.

Definition 3 (MSC)**.** *The message sequence chart $msc(e)$ associated with a sequence of actions $e = a_1 \dots a_n$ is a tuple $(Ev, \lambda, \prec)$, where*

- *$Ev = [1..n]$ is the set of events*
- *$\lambda : Ev \to S \cup R$ tags each event with its action, i.e., $\lambda(i) = a_i$*
- *$\prec$ is defined as the transitive closure of $po \cup src$.*

We identify MSCs up to graph isomorphism (i.e., we view a MSC as a labeled graph). We write $asTr(\mathfrak{S})$ to denote the set $\{msc(e) \mid e \in asEx(\mathfrak{S})\}$ of MSCs of system $\mathfrak{S}$.

Mailbox communication imposes a number of constraints on what and when messages can be read. For instance: if two messages are sent to a same process, they will be received in the same order as they have been sent. Unmatched messages also impose some constraints: if a process p sends an unmatched message to q, it will not be able to send matched messages to q afterwards



(a) **(b)** **(c)** **(d)**

Figure 1: (a) and (b) : two MSCs that violate causal delivery: orphan message $\mathbf{v}_1$ prevents process r from receiving message $\mathbf{v}_2$ that has been sent afterwards. (c) and (d) : the MSC and the conflict graph associated with $e = send(q, p, \mathbf{v}_1) \cdot send(q, r, \mathbf{v}_3) \cdot rec(q, p, \mathbf{v}_1) \cdot send(p, q, \mathbf{v}_2) \cdot rec(p, q, \mathbf{v}_2) \cdot rec(q, r, \mathbf{v}_3)$.

(see Figure 1a); or similarly, if a process p sends an unmatched message to q, any process r that receives subsequent messages from p will not be able to send matched messages to q afterwards (see Figure 1b). When a sequence of actions satisfies the constraint imposed by mailbox communication, we say that it satisfies causal delivery. Notice that, by construction, all executions satisfy causal delivery. More precisely:

Definition 4 (Causal delivery)**.** *Let $e = a_1 \dots a_n$ be a sequence of actions, and po, src, and $\prec$ defined as above. We say that e satisfies causal delivery if there is a total order $\ll$ that contains $\prec$ such that for any two send actions $a_i = send(p, q, \mathbf{v})$, $a_j = send(p', q', \mathbf{v}') \in S$ such that $i \ll j$ and $q = q'$, either a_j is unmatched, or there are i', j' such that $a_i \vdash\!\dashv a_{i'}$, $a_j \vdash\!\dashv a_{j'}$, and $i' \ll j'$.*

We recall from [4] the definition of *conflict graph* depicting the causal dependencies between message exchanges. Intuitively, we have a dependency whenever two messages share a common process. For instance a $\xrightarrow{SS}$ dependency between messages v and v' expresses the fact that v' has been sent after v by the same process.

Definition 5 (Conflict graph)**.** *The conflict graph $\mathsf{CG}(e)$ of a sequence of actions $e = a_1 \cdots a_n$ is the labeled graph $(V, \{\xrightarrow{XY}\}_{X,Y \in \{R,S\}})$ where V is the set of message exchanges of e, and for all $X, Y \in \{S, R\}$, for all $v, v' \in V$, there is a XY dependency edge $v \xrightarrow{XY} v'$ between v and v' if there are $i < j$ such that $\{a_i\} = v \cap X$, $\{a_j\} = v' \cap Y$, and $\mathsf{proc}_X(v) = \mathsf{proc}_Y(v')$.*

Figures 1c and 1d illustrate the MSC associated with an execution together with its conflict graph. We write $v \to v'$ if $v \xrightarrow{XY} v'$ for some $X, Y \in \{R, S\}$, and $v \to^* v'$ if there is a (possibly empty) path from v to v'.

3 k-synchronous and k-synchronizable executions

In this section, we define k-synchronous and k-synchronizable executions and we give a characterisation of k-synchronizable executions based on their conflict graph, correcting an error in Theorem 1 in [4]. In the rest of the paper, k denotes a fixed integer $k \geq 1$. A k-exchange is a sequence of actions starting with at most k sends and followed by at most k receives matching some of the sends.

A *k-synchronous* execution is a sequence of *k-exchanges*, such that a message sent during a k-exchange cannot be received during a subsequent one: either it is received during the same k-exchange, or it remains orphan forever.

Definition 6 (k-synchronous)**.** *A sequence of actions e is k-synchronous if there are $e_1, \ldots, e_n$ such that $e = e_1 \cdot e_2 \cdots e_n$ and*

1. *for all $i \in [1..n]$, $e_i \in S^{\leq k} \cdot R^{\leq k}$,*
2. *e satisfies causal delivery,*
3. *for all j, j' such that $a_j \vdash\!\dashv a_{j'}$ holds in e, $a_j \vdash\!\dashv a_{j'}$ holds in some e_i.*

A MSC $msc(e)$ is k-synchronous if there is a k-synchronous execution e' such that $msc(e) = msc(e')$. A sequence of actions e is k-synchronizable if there is a k-synchronous execution e' such that $msc(e) = msc(e')$.

Example 7 (k-synchronous executions)**.**

1. *Execution $e = send(p, q, \mathbf{v}_1) \cdot send(p', q', \mathbf{v}_2) \cdot rec(p, q, \mathbf{v}_1) \cdot rec(p', q', \mathbf{v}_2)$ is 2-synchronous. Its associated MSC $msc(e)$ is 1-synchronous, as $msc(e) = msc(e')$ with $e' = send(p, q, \mathbf{v}_1) \cdot rec(p, q, \mathbf{v}_1) \cdot send(p', q', \mathbf{v}_2) \cdot rec(p', q', \mathbf{v}_2)$. In other words, e is 1-synchronizable.*
2. *The MSC in Figure 2a is not k-synchronous for any k. All messages must be grouped in the same k-exchange, but it is not possible to schedule all the sends first, because the reception of $\mathbf{v}_1$ happens before the sending of $\mathbf{v}_3$. Still, this MSC satisfies causal delivery.*
3. *The MSC depicted in Figure 2b is 1-synchronous. This is the only way to chop this MSC in 1-exchanges, it would not be possible for instance to place $\mathbf{v}_3$ in a 1-exchange before $\mathbf{v}_1$. Note, also, that this MSC satisfies causal delivery, but $\mathbf{v}_3$ must be sent before $\mathbf{v}_1$.*

Following standard terminology, we say that a subset $U \subseteq V$ of vertices is a *strongly connected component* (SCC) of a given graph $(V, \to)$ if between any two vertices $v, v' \in U$, there exist two oriented paths $v \to^* v'$ and $v' \to^* v$. The statement below fixes some issues with Theorem 1 in [4] (see Section 6 for a detailed discussion).

Theorem 8 (Graphical characterisation of k-synchronizable executions)**.** *Let e be a sequence of actions that satisfies causal delivery. Then $msc(e)$ is k-synchronous iff every SCC in its conflict graph is of size at most k and if no RS edge occurs on any cyclic path.*

Proof. Let e be an execution of the system:

$\Longrightarrow$ If e is k-synchronous, then $e = e_1 \cdots e_n$ where each e_i is a k-exchange. For every vertex v of the conflict graph $\mathsf{CG}(e)$, there is exactly one index $\iota(v) \in [1..n]$ such that $v \subseteq e_{\iota(v)}$. Now, observe that if there is an edge from v to v' in the conflict graph, some action of v must happen before some action of v', i.e., $\iota(v) \leq \iota(v')$. So if v, v' are on a same SCC, $\iota(v) = \iota(v')$, they must both occur within the same k-exchange. Since each k-exchange contains at most k message exchanges, this shows that all SCC are of size at most k. Observe also that if

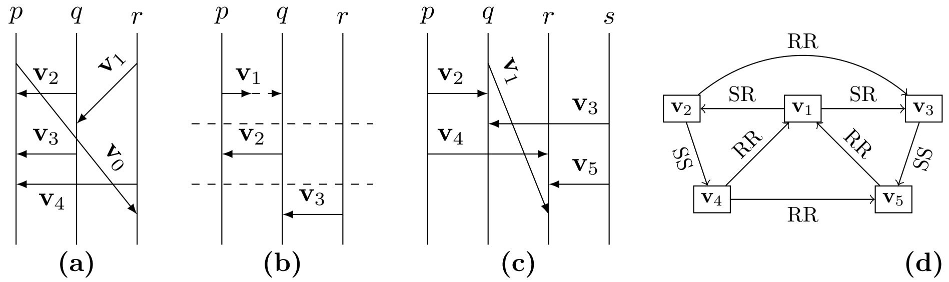


Figure 2: (a) the MSC of Example 7.2. (b) the MSC of Example 7.3. (c) MSC $msc(e)$ and (d) its conflict graph, where e is the execution $e = send(p,q,\mathbf{v}_2) \cdot send(s,q,\mathbf{v}_3) \cdot send(p,r,\mathbf{v}_4) \cdot send(s,r,\mathbf{v}_5) \cdot send(q,r,\mathbf{v}_1) \cdot rec(p,q,\mathbf{v}_2) \cdot rec(s,q,\mathbf{v}_3) \cdot rec(p,r,\mathbf{v}_4) \cdot rec(s,r,\mathbf{v}_5) \cdot rec(q,r,\mathbf{v}_1)$

$v \xrightarrow{RS} v'$, then $\iota(v) < \iota(v')$, since within a k-exchange all the sends precede all the receives. So an RS edge cannot occur on a cyclic path.

$\Longleftarrow$ Assume now that conflict graph $\mathsf{CG}(e)$ neither contains a SCC of size greater than $k+1$ nor a cyclic path with an RS edge. Let $V_1, \ldots, V_n$ be the set of maximal SCCs of the conflict graph, listed in some topological order. For a fixed i, let $e_i = s_1 \ldots s_m r_1 \ldots r_{m'}$ be the enumeration of the actions of the message exchanges of V_i defined by taking first all send actions of V_i in the order in which they appear in e, and second all the receive actions of V_i in the same order as in e. Let $e' = e_1 \ldots e_n$. Then $\mathsf{CG}(e')$ is the same as $\mathsf{CG}(e)$: indeed, the permutation of actions we defined could only postpone a receive after a send of a same SCC, therefore it could only replace some $v \xrightarrow{RS} v'$ edge with an $v' \xrightarrow{SR} v$ edge between two vertices v, v' of a same SCC, but we assumed that the SCCs do not contain RS edges, so it does not happen. Therefore e and e' have the same conflict graph, and $msc(e') = msc(e)$. Moreover, also by hypothesis, $|V_i| \leq k$ for all i, therefore each e_i is a k-exchange, and finally e' is k-synchronous. □

Example 9 (A 5-synchronous MSC)**.** *Figure 2c depicts a* 5*-synchronous MSC that is not* 4*-synchronous. Indeed, its conflict graph (Figure 2d) contains a SCC of size 5 (all vertices are on the same SCC).*

4 Decidability of reachability for k-synchronizable systems

We show, now, that the reachability problem is decidable for k-synchronizable systems. While proving this result, we have to face several non-trivial aspects of causal delivery that were missed in [4] and that require a completely new approach.

We write $sTr_k(\mathfrak{S})$ to denote the set

$$\{msc(e) \mid e \in asEx(\mathfrak{S}) \text{ and } msc(e) \text{ is } k\text{-synchronous}\}.$$

Definition 10 (k-synchronizable system)**.** *A system* $\mathfrak{S}$ *is* k-synchronizable *if all its executions are k-synchronizable, i.e.,* $sTr_k(\mathfrak{S}) = asTr(\mathfrak{S})$*.*

$$\text{(Rule 1)}\ \frac{\mathbf{v}_1 \xrightarrow{XY} \mathbf{v}_2}{\mathbf{v}_1 \overset{XY}{\dashrightarrow} \mathbf{v}_2} \qquad \text{(Rule 2)}\ \frac{\mathbf{v} \text{ is matched}}{\mathbf{v} \overset{SR}{\dashrightarrow} \mathbf{v}} \qquad \text{(Rule 3)}\ \frac{\mathbf{v}_1 \xrightarrow{RR} \mathbf{v}_2}{\mathbf{v}_1 \overset{SS}{\dashrightarrow} \mathbf{v}_2}$$

$$\text{(Rule 4)}\ \frac{\begin{array}{c}\mathbf{v}_1 \text{ is matched} \qquad \mathbf{v}_2 \text{ is unmatched} \\ \mathsf{proc}_R(\mathbf{v}_1) = \mathsf{proc}_R(\mathbf{v}_2)\end{array}}{\mathbf{v}_1 \overset{SS}{\dashrightarrow} \mathbf{v}_2} \qquad \text{(Rule 5)}\ \frac{\mathbf{v}_1 \overset{XY}{\dashrightarrow}\overset{YZ}{\dashrightarrow} \mathbf{v}_2}{\mathbf{v}_1 \overset{XZ}{\dashrightarrow} \mathbf{v}_2}$$

Figure 3: Deduction rules for extended dependency edges of the conflict graph

In other words, a system $\mathfrak{S}$ is k-synchronizable if for every execution e of $\mathfrak{S}$, there exists another execution e' of $\mathfrak{S}$ such that $msc(e) = msc(e')$ and e' is k-synchronous. In particular, a system may be k-synchronizable even if some of its executions fill the buffers with more than k messages. For a k-synchronizable system, the reachability problem reduces to the reachability through a k-synchronous execution. In order to show that k-synchronous reachability is decidable, we establish that the set of k-synchronous executions is regular. More precisely, we want to define a finite state automaton that accepts a sequence $e_1 \cdot e_2 \cdots e_n$ of k-exchanges if and only if it satisfies causal delivery.

We start by giving a graphical characterisation of causal delivery. For this, we define the *extended edges* $v \overset{XY}{\dashrightarrow} v'$ of a given conflict graph. The relation $\overset{XY}{\dashrightarrow}$ is defined in Figure 3 with $X, Y \in \{S, R\}$. Intuitively, $v \overset{XY}{\dashrightarrow} v'$ expresses that event X of v must happen before event Y of v' due to either their order on the same machine (Rule 1), or the fact that a send happens before its matching receive (Rule 2), or due to the mailbox semantics (Rules 3 and 4), or because of a chain of such dependencies (Rule 5). We observe that in the extended conflict graph, obtained applying such rules, a cyclic dependency appears whenever causal delivery is not satisfied.

Example 11. *Figures 5a and 5b depict a MSC that does not verify causal delivery together with its associated conflict graph with some extended edges. Notice that there is a cyclic dependency* $\mathbf{v}_1 \overset{SS}{\dashrightarrow} \mathbf{v}_1$. *This is the sign that this MSC violates causal delivery.*

Theorem 12 (Graphical characterisation of causal delivery)**.** *A sequence of actions e satisfies causal delivery iff there is no cyclic causal dependency of the form $v \overset{SS}{\dashrightarrow} v$ for some vertex v of the associated extended conflict graph.*

Proof. $\Rightarrow$ Assume that $msc(e)$ satisfies causal delivery. Then there is a total order $\ll$ on the events that is a linearisation of $\prec = (po \cup src)^+$ (cfr. Definition 3) with the property stated in Definition 4. We claim that if $v \overset{XY}{\dashrightarrow} v'$, $\{a_i\} = v \cap X$ and $\{a_j\} = v' \cap Y$, then $i \ll j$. The proof of this claim is by induction on the derivation tree of $v \overset{XY}{\dashrightarrow} v'$:

- case of Rule 1 : $(i, j) \in po$, so $i \ll j$;
- case of Rule 2 : $(i, j) \in src$, so $i \ll j$;
- cases of Rules 3 and 4 : by definition of causal delivery;
- case of Rule 5 : there is v_3 such that $v_1 \overset{XZ}{\dashrightarrow} v_3 \xrightarrow{ZY} v_2$. Let a_l be the Z action of v_3. By inductive hypothesis, $i \ll l \ll j$, and by transitivity of $\ll$, $i \ll j$.

So we proved our claim, and $\ll$ extends $\overset{XY}{\dashrightarrow}$. As a consequence, there is no $\overset{SS}{\dashrightarrow}$ cycle.

$\Leftarrow$ Assume that the extended dependency graph does not contain any $\overset{SS}{\dashrightarrow}$ cycle. Let us first show that it does not contain any $\overset{RR}{\dashrightarrow}$ cycle either. By contradiction assume there is some v such that $v \overset{RR}{\dashrightarrow} v$. Since there is no $\overset{SS}{\dashrightarrow}$ cycle, there is no v' on the cyclic path such that $v \overset{RS}{\dashrightarrow} v' \overset{SR}{\dashrightarrow} v$. So $v(\xrightarrow{RR})^*v$, and we reach a contradiction, as $\xrightarrow{RR}$ is included in *po* which is acyclic. So $\overset{RR}{\dashrightarrow}$ is acyclic, and $\overset{XY}{\dashrightarrow}$ defines a partial order on actions. Let us pick some linearisation of that order, and let $\ll$ denote the associated order on indexes, i.e., $\ll$ is a total order such that for any X action $a_i \in v_i$ and Y action $a_j \in v_j$, $v_i \overset{XY}{\dashrightarrow} v_j$ implies $i \ll j$. We want to show that $\ll$ satisfies the property of Definition 4. Let $i \ll j$ with $a_i, a_j \in S$ and $\mathsf{proc}_R(a_i) = \mathsf{proc}_R(a_j)$, and let v_i, v_j be the two vertices such that $a_i \in v_i$ and $a_j \in v_j$. Since $\ll$ extends $\overset{XY}{\dashrightarrow}$, either $v_i \overset{SS}{\dashrightarrow} v_j$ or $\neg(v_j \overset{SS}{\dashrightarrow} v_i)$.

- Assume that $v_i \overset{SS}{\dashrightarrow} v_j$. If v_i is unmatched, then v_j must be unmatched otherwise by Rule 4 we would have $v_j \overset{SS}{\dashrightarrow} v_i$, which would violate the acyclicity hypothesis. On the other hand, if both v_i and v_j are matched, then $v_i \xrightarrow{RR} v_j$, otherwise we would have $v_j \xrightarrow{RR} v_j$ and by Rule 3 $v_j \overset{SS}{\dashrightarrow} v_j$, which would violate the acyclicity hypothesis. So there are i', j' such that $v_i = \{a_i, a_{i'}\}$, $v_j = \{a_j, a_{j'}\}$ and $i' \ll j'$, as required by Definition 4.

- Assume that $\neg(v_i \overset{SS}{\dashrightarrow} v_j)$ and $\neg(v_j \overset{SS}{\dashrightarrow} v_i)$. Then both sends are unmatched (because of Rules 3 and 4), therefore the property of Definition 4 holds, concluding the proof.

□

Let us now come back to our initial problem: we want to recognise with a finite memory the sequences $e_1 \cdot e_2 \cdots e_n$ of k-exchanges that satisfy causal delivery. We proceed by reading each k-exchange one by one in sequence. This entails that we have only a partial view of the conflict graph of the whole sequence, but we want to determine whether the acyclicity condition of Theorem 12 is satisfied in the whole conflict graph. The crucial observation is that the only edges that may "go back in time" are those generated by Rule 4. This means that we have to remember enough information from the previously examined k-exchanges to determine whether the current k-exchange contains a vertex v that shares an edge with some unmatched vertex v' seen in a previous k-exchange and whether this could participate in a cycle. This is achieved by computing two sets of processes $C_{S,p}$ and $C_{R,p}$ that collect the following information: a process q is in $C_{S,p}$ if it performs a send action causally after an unmatched send to p, or it is the sender of the unmatched send; a process q belongs to $C_{R,p}$ if it receives a message that was sent after some unmatched message directed to p. Thus, if we assume that $\mathsf{Unmatched}_p$ is the set of the unmatched sends to p, we have:

$$C_{S,p} = \{\mathsf{proc}_S(v) \mid v' \overset{SS}{\dashrightarrow} v \ \&\ v' \in \mathsf{Unmatched}_p \ \&\ \mathsf{proc}_R(v') = p\}$$

$$C_{R,p} = \{\mathsf{proc}_R(v) \mid v' \overset{SS}{\dashrightarrow} v \ \&\ v' \in \mathsf{Unmatched}_p \ \&\ \mathsf{proc}_R(v') = p \ \&\ v \cap R \neq \emptyset\}$$

These sets abstract and carry from one k-exchange to another the necessary information to detect violations of causal delivery. We want to compute them in any local conflict graph of a k-exchange incrementally, i.e., knowing what they were at the end of the previous k-exchange, we want to compute them at the end of the current one. More precisely, let $e = s_1 \cdots s_m \cdot r_1 \cdots r_{m'}$ be a k-exchange, $\mathsf{CG}(e) = (V, E)$ the conflict graph of e and $B : \mathbb{P} \to (2^{\mathbb{P}} \times 2^{\mathbb{P}})$ associates to each $p \in \mathbb{P}$ the two sets $B(p) = (C_{S,p}, C_{R,p})$. Then, the conflict graph $\mathsf{CG}(e, B)$ is the graph (V', E') with $V' = V \cup \{\psi_p \mid p \in \mathbb{P}\}$ and $E' \supseteq E$ as defined below. For each process $p \in \mathbb{P}$, the "summary node" ψ_p shall account for all past unmatched messages sent to p that occurred in some k-exchange before e. E' is the set E of edges $\xrightarrow{XY}$ among message exchanges of e, as in Definition 5, augmented with the following set of extra edges taking into account the summary nodes.

$$\begin{aligned}
&\{\psi_p \xrightarrow{SX} v \mid \mathsf{proc}_X(v) \in C_{S,p} \;\&\; v \cap X \neq \emptyset \text{ for some } X \in \{S, R\}\} && (1)\\
\cup\; &\{\psi_p \xrightarrow{SS} v \mid \mathsf{proc}_X(v) \in C_{R,p} \;\&\; v \cap R \neq \emptyset \text{ for some } X \in \{S, R\}\} && (2)\\
\cup\; &\{\psi_p \xrightarrow{SS} v \mid \mathsf{proc}_R(v) \in C_{R,p} \;\&\; v \text{ is unmatched}\} && (3)\\
\cup\; &\{v \xrightarrow{SS} \psi_p \mid \mathsf{proc}_R(v) = p \;\&\; v \cap R \neq \emptyset\} \;\cup\; \{\psi_q \xrightarrow{SS} \psi_p \mid p \in C_{R,q}\} && (4)
\end{aligned}$$

These extra edges summarise/abstract the connections to and from previous k-exchanges. Equation (1) considers connections $\xrightarrow{SS}$ and $\xrightarrow{SR}$ that are due to two sends messages or, respectively, a send and a receive on the same process. Equations (2) and (3) considers connections $\xrightarrow{RR}$ and $\xrightarrow{RS}$ that are due to two received messages or, respectively, a receive and a subsequent send on the same process. Notice how the rules in Figure 3 would then imply the existence of a connection $\overset{SS}{\dashrightarrow}$, in particular Equation (3) abstract the existence of an edge $\overset{SS}{\dashrightarrow}$ built because of Rule 4. Equations in (4) abstract edges that would connect the current k-exchange to previous ones. As before those edges in the global conflict graph would correspond to extended edges added because of Rule 4 in Figure 3. Once we have this enriched local view of the conflict graph, we take its extended version. Let $\overset{XY}{\dashrightarrow}$ denote the edges of the extended conflict graph as defined from rules in Figure 3 taking into account the new vertices ψ_p and their edges.

Finally, let $\xRightarrow[\mathsf{cd}]{e,k}$ be the transition relation given in Figure 4 among abstract configurations of the form $(\vec{l}, B)$ where $\vec{l}$ is a global control state of the system and $B : \mathbb{P} \to \left(2^{\mathbb{P}} \times 2^{\mathbb{P}}\right)$ associates to each process p a pair of sets of processes $B(p) = (C_{S,p}, C_{R,p})$. Transition $\xRightarrow[\mathsf{cd}]{e,k}$ updates these sets with respect to the current k-exchange e. Causal delivery is verified by checking that for all $p \in \mathbb{P}, p \notin C'_{R,p}$ meaning that there is no cyclic dependency as stated in Theorem 12. The initial state is $(\vec{l_0}, B_0)$, where $B_0 : \mathbb{P} \to (2^{\mathbb{P}} \times 2^{\mathbb{P}})$ denotes the function such that $B_0(p) = (\emptyset, \emptyset)$ for all $p \in \mathbb{P}$.

Example 13 (An invalid execution)**.** *Let $e = e_1 \cdot e_2$ with e_1 and e_2 2-exchanges of this execution such that $e_1 = \mathit{send}(q, r, \mathbf{v}_1) \cdot \mathit{send}(q, s, \mathbf{v}_2) \cdot \mathit{rec}(q, s, \mathbf{v}_2)$ and $e_2 = \mathit{send}(p, s, \mathbf{v}_3) \cdot \mathit{rec}(p, s, \mathbf{v}_3) \cdot \mathit{send}(p, r, \mathbf{v}_4) \cdot \mathit{rec}(p, r, \mathbf{v}_4)$. Figures 5a and 5c show the MSC and corresponding conflict graph of each of the 2-exchanges.*

$$\frac{\begin{array}{c} e = s_1 \cdots s_m \cdot r_1 \cdots r_{m'} \qquad s_1 \cdots s_m \in S^* \qquad r_1 \cdots r_{m'} \in R^* \qquad 0 \le m' \le m \le k \\ (\vec{l}, \mathtt{Buf}_0) \overset{e}{\Rightarrow} (\vec{l'}, \mathtt{Buf}) \text{ for some } \mathtt{Buf} \\ \text{for all } p \in \mathbb{P} \quad B(p) = (C_{S,p}, C_{R,p}) \text{ and } B'(p) = (C'_{S,p}, C'_{R,p}), \\ \mathsf{Unmat}_p = \{\psi_p\} \cup \{v \mid v \text{ is unmatched } \& \ \mathsf{proc}_R(v) = p\} \\ C'_{X,p} = C_{X,p} \cup \{p \mid p \in C_{X,q} \ \& \ v \overset{SS}{\dashrightarrow} \psi_q \ \& \ (\mathsf{proc}_R(v) = \pi \text{ or } v = \psi_{\mathsf{start}})\} \\ \cup \ \{\mathsf{proc}_X(v) \mid v \in \mathsf{Unmat}_p \cap V \ \& \ X = S\} \\ \cup \{\mathsf{proc}_X(v') \mid v \overset{SS}{\dashrightarrow} v' \ \& \ v \in \mathsf{Unmat}_p \ \& \ v \cap X \neq \emptyset\} \\ \text{for all } p \in \mathbb{P}, p \notin C'_{R,p} \end{array}}{(\vec{l}, B) \xRightarrow[\mathsf{cd}]{e,k} (\vec{l'}, B')}$$

Figure 4: Definition of the relation $\xRightarrow[\mathsf{cd}]{e,k}$

Note that two edges of the global graph (in blue) "go across" k-exchanges. These edges do not belong to the local conflict graphs and are mimicked by the incoming and outgoing edges of summary nodes. The values of sets $C_{S,r}$ and $C_{R,r}$ at the beginning and at the end of the k-exchange are given on the right. All other sets $C_{S,p}$ and $C_{R,p}$ for $p \neq r$ are empty, since there is only an unmatched message to process r. Notice how at the end of the second k-exchange, $r \in C_{R,r}$ signalling that message v_4 violates causal delivery.

Next lemma proves that the rule in Figure 4 properly characterises causal delivery.

Lemma 14. *A sequence of actions e is a k-synchronous execution iff $e = e_1 \cdots e_n$ such that $(\vec{l_0}, B_0) \xRightarrow[\mathsf{cd}]{e_1,k} \cdots \xRightarrow[\mathsf{cd}]{e_n,k} (\vec{l'}, B')$ for some global state $\vec{l'}$ and some $B' : \mathbb{P} \to (2^{\mathbb{P}} \times 2^{\mathbb{P}})$.*

Proof. $\Rightarrow$ Since e is k-synchronous then $e = e_1 \cdots e_n$. The proof proceeds by induction on n.

Base case If $n = 1$ then $e = e_1$. Thus there is only one k-exchange and the local conflict graph $\mathsf{CG}(e, B)$ is the same as the complete global one $\mathsf{CG}(e)$. By hypothesis, as e is an execution in $\mathfrak{S}$, we have that for some $\mathtt{Buf}$, $(\vec{l}, \mathtt{Buf}_0) \xRightarrow[\mathsf{cd}]{e,} (\vec{l'}, \mathtt{Buf})$.

By contradiction, suppose that $\exists p \in \mathbb{P}$ such that $p \in C'_{R,p}$. Whence there exists v' matched, such that $p = \mathsf{proc}_R(v')$ and $v \overset{SS}{\dashrightarrow} v'$ with $v \in \mathsf{Unmat}_p$. By Rule 4 (Figure 3), an edge $v' \overset{SS}{\dashrightarrow} v$ has been added to the extended conflict graph. Thus, there is a cycle $\overset{SS}{\dashrightarrow}$ from v to v and this violates Theorem 12, which is a contradiction.

Inductive step If $n > 1$, by inductive hypothesis, we have $(\vec{l_0}, B_0) \xRightarrow[\mathsf{cd}]{e_1,k} \cdots \xRightarrow[\mathsf{cd}]{e_{n-1},k} (\vec{l_{n-1}}, B)$, with $B = (C_{S,p}, C_{R,p})_{p \in \mathbb{P}}$. By inductive hypothesis we have that

$$C_{S,p} = \{\mathsf{proc}_S(v') \mid v \overset{SS}{\dashrightarrow} v' \ \& \ v \text{ not matched } \& \ \mathsf{proc}_S(v) = p\}$$

$$C_{R,p} = \{\mathsf{proc}_R(v') \mid v \overset{SS}{\dashrightarrow} v' \ \& \ v \text{ not matched } \& \ \mathsf{proc}_R(v) = p \ \& \ v' \cap R \neq \emptyset\}$$

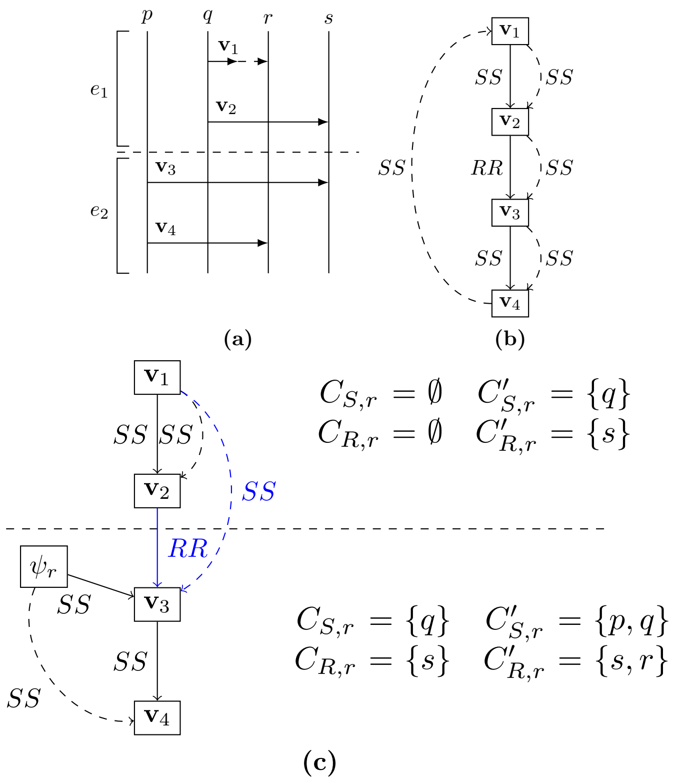

p
q
r
s
e1
e2
v1
v2
v3
v4
SS
RR
(a)
(b)
ψr
$C_{S,r} = \emptyset \quad C'_{S,r} = \{q\}$
$C_{R,r} = \emptyset \quad C'_{R,r} = \{s\}$
$C_{S,r} = \{q\} \quad C'_{S,r} = \{p,q\}$
$C_{R,r} = \{s\} \quad C'_{R,r} = \{s,r\}$
(c)

Figure 5: (a) MSC of execution e, (b) the associated global conflict graph, (c) the conflict graphs of its k-exchanges

By contradiction, suppose that there is a process $p \in C'_{R,p}$. Then by construction there exist two nodes v and v' such that $v \overset{SS}{\dashrightarrow} v'$, $v \in \mathsf{Unmat}_p$, v' matched and $\mathsf{proc}_R(v') = p$. We can have the following situations:

1. $v \in V$, then both message exchanges v and v' with v unmatched and v' matched are in the current k-exchange then we can easily reach a contradiction and the proof proceeds as in the base case.
2. $v = \psi_p$, then by inductive hypothesis there exists a non-matched message $v_p \in V$ belonging to a previous k-exchange. We want to show that if this is the case we can reconstruct a cyclic path in the extended conflict graph of the execution e, which is a contradiction. We assume that by inductive hypothesis $\mathsf{CG}(e)$ has been reconstructed from the local conflict graphs considering actions in $e_1 \dots e_{n-1}$. We now analyse the last k-exchange and describe to what each edge corresponds in $\mathsf{CG}(e)$. There are four cases:

   (a) $v_1 \xrightarrow{XY} v_2$ with $v_1, v_2 \in V$, this edge exists also in $\mathsf{CG}(e)$

   (b) $\psi_q \xrightarrow{SX} v_1$ with $v_1 \in V$. Then in $\mathsf{CG}(e)$ there exists an unmatched message v_q and this extra edge has been constructed from Equations 1, 2 or 3:
   If $\psi_q \xrightarrow{SR} v_1$ then $\mathsf{proc}_R(v_1) \in C_{S,q}$ thus by inductive hypothesis there exists $v \in V$ in $\mathsf{CG}(e)$ such that $v_q \overset{SS}{\dashrightarrow} v$ and $\mathsf{proc}_S(v) =$

$\mathsf{proc}_R(v_1)$. Whence there exists an edge $v \overset{SR}{\dashrightarrow} v_1$ in $\mathsf{CG}(e)$. If the edge $\psi_q \xrightarrow{SS} v_1$ has been added as $\mathsf{proc}_S(v_1) \in C_{S,q}$ then by inductive hypothesis there exists, in $\mathsf{CG}(e)$, a node v reachable with an edge $\overset{SS}{\dashrightarrow}$ from v_q such that $\mathsf{proc}_S(v) = \mathsf{proc}_S(v_1)$. Thus an edge $v \overset{SS}{\dashrightarrow} v_1$ exists in $\mathsf{CG}(e)$.

If the edge $\psi_q \xrightarrow{SS} v_1$ has been added as $\mathsf{proc}_R(v_1) \in C_{R,q}$ and v_1 is a matched send. Then by inductive hypothesis there exists, in $\mathsf{CG}(e)$, a node v reachable with an edge $\overset{SS}{\dashrightarrow}$ from v such that $\mathsf{proc}_R(v) = \mathsf{proc}_R(v_1)$. Whence in $\mathsf{CG}(e)$ there exists an edge $v \xrightarrow{RR} v_1$.

If the edge $\psi_q \xrightarrow{SS} v_1$ has been added as $\mathsf{proc}_R(v_1) \in C_{R,q}$ and v_1 is an unmatched send. Then by inductive hypothesis there exists, in $\mathsf{CG}(e)$, a node v reachable with an edge $\overset{SS}{\dashrightarrow}$ from v such that $\mathsf{proc}_R(v) = \mathsf{proc}_R(v_1)$. Whence in $\mathsf{CG}(e)$, because of Rule (4) in Figure 3 there exists an edge $v \overset{SS}{\dashrightarrow} v_1$.

If the edge $\psi_q \xrightarrow{SS} v_1$ has been added as $\mathsf{proc}_S(v_1) \in C_{R,q}$. Then by inductive hypothesis there exists, in $\mathsf{CG}(e)$, a node v reachable with an edge $\overset{SS}{\dashrightarrow}$ from v such that $\mathsf{proc}_R(v) = \mathsf{proc}_S(v_1)$. Whence in $\mathsf{CG}(e)$, there exists an edge $v \xrightarrow{RS} v_1$.

(c) $v_1 \xrightarrow{SS} \psi_q$ with $v_1 \in V$ and v_1 matched, then we know $\mathsf{proc}_R(v_1) = q$ and because of Rule (4) in Figure 3 in $\mathsf{CG}(e)$ there exists an edge $v_1 \overset{SS}{\dashrightarrow} v_q$.

(d) $\psi_q \xrightarrow{SS} \psi_r$, thus $r \in C_{R,q}$. This means that there exists a matched message v such that $v_q \xrightarrow{SS} v$ and $\mathsf{proc}_R(v) = r$. Thus, in $\mathsf{CG}(e)$, we can add, because of Rule (4) in Figure 3, the edge $v \overset{SS}{\dashrightarrow} v_r$.

Then it follows that if there exists an edge $\psi_p \overset{SS}{\dashrightarrow} v'$ it means that an edge $v_p \overset{SS}{\dashrightarrow} v'$ exists in the global extended conflict graph and thus by applying Rule (4) in Figure 3 we can reach a contradiction, as we have a cycle.

$\Leftarrow$ If $e = e_1 \cdots e_n$, where each e_i corresponds to a valid k-exchange. Suppose by contradiction that e is not an execution, thus there exists a message that violates causal delivery. By Theorem 12 then the global extended conflict graph must contain an edge $v \overset{SS}{\dashrightarrow} v$. This means that there is an unmatched message v_p to process p that is causally followed by a matched message v to the same process p. Since each e_i is a valid k-exchange we know that such an edge cannot appear in any of the local conflict graphs. Indeed, if such an edge existed then there should be an edge $\overset{SS}{\dashrightarrow}$ from v_p or ψ_p (if the two messages belong to two different k-exchanges) to v. But in this case we would have $p \in C_{R,P}$ which is a contradiction. □

Note that there are only finitely many abstract configurations of the form $(\vec{l}, B)$ with $\vec{l}$ a tuple of control states and $B : \mathbb{P} \to (2^{\mathbb{P}} \times 2^{\mathbb{P}})$. Therefore $\overset{e,k}{\underset{\mathsf{cd}}{\Longrightarrow}}$

is a relation on a finite set, and the set $sTr_k(\mathfrak{S})$ of k-synchronous executions of a system $\mathfrak{S}$ forms a regular language. It follows that it is decidable whether a given abstract configuration of the form $(\vec{l}, B)$ is reachable from the initial configuration following a k-synchronous execution. This is the content of next theorem.

Theorem 15. *Let $\mathfrak{S}$ be a k-synchronizable system and $\vec{l}$ a global control state of $\mathfrak{S}$. The problem whether there exists $e \in asEx(\mathfrak{S})$ and* $\texttt{Buf}$ *such that $(\vec{l_0}, \texttt{Buf}_0) \overset{e}{\Rightarrow} (\vec{l}, \texttt{Buf})$ is decidable.*

Remark 16. *Deadlock-freedom, unspecified receptions, and absence of orphan messages are other properties that become decidable for a k-synchronizable system because of the regularity of the set of k-synchronous executions.*

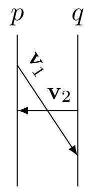
5 Decidability of k-synchronizability for mailbox systems

We establish, here, the decidability of k-synchronizability; our approach is similar to the one of Bouajjani *et al.* based on the notion of borderline violation, but we adjust it to adapt to the new characterisation of k-synchronizable executions (Theorem 8).

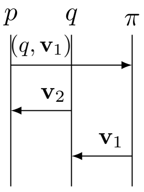
Definition 17 (Borderline violation)**.** *A non k-synchronizable execution e is a borderline violation if $e = e' \cdot r$ and e' is k-synchronizable.*

Note that a system $\mathfrak{S}$ that is not k-synchronizable always admits at least one borderline violation $e' \cdot r \in asEx(\mathfrak{S})$ with $r \in R$: indeed, there is at least one execution $e \in asEx(\mathfrak{S})$ that is not k-synchronizable, which contains a unique minimal prefix of the form $e' \cdot r$ that is not k-synchronizable; moreover since e' is k-synchronizable, r cannot be a k-exchange of just one send action, therefore it must be a receive action. In order to find such a borderline violation, Bouajjani *et al.* introduced an instrumented system $\mathfrak{S}'$ that behaves like $\mathfrak{S}$, except that it contains an extra process π, and that non-deterministically a message that should have been sent from a process p to a process q may now be sent from p to π, and later forwarded by π to q. In $\mathfrak{S}'$, each process p has the possibility, instead of sending a message $\mathbf{v}$ to q, to deviate this message to π; if it does so, p continues its execution as if it really had sent it to q. Note also that the message sent to π get tagged with the original destination process q. Similarly, for each possible reception, a process has the possibility to receive a given message not from the initial sender but from π. The process π has an initial state from which it can receive any messages from the system. Each reception makes it go into a different state. From this state, it is able to send the message back to the original recipient. Once a message is forwarded, π reaches its final state and remains idle. The following example illustrates this situation.

Example 18 (A deviated message). *e_1 is not 1-synchronous. It is indeed borderline in $\mathfrak{S}$ as if we delete the last reception, it becomes 1-synchronous. In $msc(e_2)$ from the instrumented system $\mathfrak{S}'$, the message $\mathbf{v}_1$ is deviated. Note that $msc(e_2)$ is 1-synchronous. In this case, the instrumented system $\mathfrak{S}'$ in the 1-synchronous semantics "reveals" the existence of a borderline violation of $\mathfrak{S}$.*



$msc(e_1)$



$msc(e_2)$

Definition 19 (Instrumented system $\mathfrak{S}'$). *Let $\mathfrak{S} = ((L_p, \delta_p, l_p^0) \mid p \in \mathbb{P})$ be a system of communicating machines. The instrumented system $\mathfrak{S}'$ associated to $\mathfrak{S}$ is defined such that $\mathfrak{S}' = \big((L_p, \delta'_p, l_p^0) \mid p \in \mathbb{P} \cup \{\pi\}\big)$ where for all $p \in \mathbb{P}$:*

$$\delta'_p = \delta_p \cup \{l_1 \xrightarrow{send(p,\pi,(q,\mathbf{v}))} l_2 \mid l_1 \xrightarrow{send(p,q,\mathbf{v})} l_2 \in \delta_p\}$$
$$\cup\{l_1 \xrightarrow{rec(\pi,p,\mathbf{v})} l_2 \mid l_1 \xrightarrow{rec(q,p,\mathbf{v})} l_2 \in \delta_p\}$$

Process π is the communicating automaton $(L_\pi, l_\pi^0, \delta_\pi)$ where

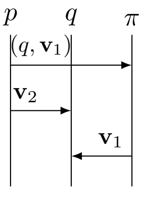
- *$L_\pi = \{l_\pi^0, l_f\} \cup \{l_{q,\mathbf{v}} \mid \mathbf{v} \in \mathbb{V}, q \in \mathbb{P}\}$, and*
- *$\delta_\pi = \{l_\pi^0 \xrightarrow{rec(p,\pi,(q,\mathbf{v}))} l_{q,\mathbf{v}} \mid send(p,q,\mathbf{v}) \in S\} \cup \{l_{q,\mathbf{v}} \xrightarrow{send(\pi,q,\mathbf{v})} l_f \mid rec(p,q,\mathbf{v}) \in R\}$.*

Thus, each message can be redirected to π non-deterministically and π is able to participate in the deviation of one and only one message. For a given execution $e \cdot r \in asEx(\mathfrak{S})$ that ends with a reception, there exists an execution $\mathsf{deviate}(e \cdot r) \in asEx(\mathfrak{S}')$ where the message exchange associated with the reception r has been deviated to π; formally, if $e \cdot r = e_1 \cdot s \cdot e_2 \cdot r$ with $r = rec(p, q, \mathbf{v})$ and $s \vdash\!\dashv r$, then

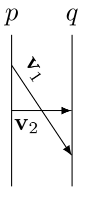
$$\mathsf{deviate}(e \cdot r) = e_1 \cdot send(p, \pi, (q, \mathbf{v})) \cdot rec(p, \pi, (q, \mathbf{v})) \cdot e_2 \cdot send(\pi, q, (\mathbf{v})) \cdot rec(\pi, q, \mathbf{v}).$$

Definition 20 (Feasible execution, bad execution). *A k-synchronizable execution e' of $\mathfrak{S}'$ is* feasible *if there is an execution $e \cdot r \in asEx(\mathfrak{S})$ such that $\mathsf{deviate}(e \cdot r) = e'$. It is* bad *if $e \cdot r$ is not k-synchronizable.*

Example 21 (A non-feasible execution). *Let e' be an execution such that $msc(e')$ is as depicted on the right. Clearly, this MSC satisfies causal delivery and could be the execution of some instrumented system $\mathfrak{S}'$. However, the sequence $e \cdot r$ such that $\mathsf{deviate}(e \cdot r) = e'$ does not satisfy causal delivery, therefore cannot be an execution of the original system $\mathfrak{S}$. In other words, the execution e' is not feasible.*



$msc(e')$



$msc(e \cdot r)$

Lemma 22. *A system $\mathfrak{S}$ is not k-synchronizable iff there is a k-synchronizable execution e' of $\mathfrak{S}'$ that is feasible and bad.*

Proof. $\Leftarrow$ Let $\mathfrak{S}$ be not k-synchronizable then there exists an execution that is not k-synchronizable which contains a unique minimal prefix of the form $e \cdot r$

with e k-synchronizable and $r = \mathit{rec}(p, q, \mathbf{v})$ a receive action. Thus there exists an $e' = \mathsf{deviate}(e \cdot r) \in \mathit{asEx}(\mathfrak{S}')$.

Since e is k-synchronizable, there exists an execution e'' such that $\mathit{msc}(e) = \mathit{msc}(e'')$ and e'' is k-synchronous. Then $e'' = e_1 \dots e_n$ and there exists a k-exchange e_i containing the send action $\mathit{send}(p, q, \mathbf{v})$. Now if we replace this action with $\mathit{send}(p, \pi, (q, \mathbf{v}))$ and we add at the end of the same k-exchange the action $\mathit{rec}(p, \pi, (q, \mathbf{v}))$. The execution in $\mathit{asEx}(\mathfrak{S}')$ remains k-synchronous. Finally if we add to e'' a new k-exchange with the actions $\mathit{send}(\pi, q, \mathbf{v})$ and $\mathit{rec}(\pi, q, \mathbf{v})$ the execution remains k-synchronous.

$\Rightarrow$ If there is a k-synchronizable execution e' of $\mathfrak{S}'$ that is feasible and bad. Then by construction $e' = \mathsf{deviate}(e \cdot r)$ and $e \cdot r$ is not k-synchronizable. Whence $\mathfrak{S}$ is not k-synchronizable and this concludes the proof.

□

As we have already noted, the set of k-synchronous executions of $\mathfrak{S}'$ is regular. The decision procedure for k-synchronizability follows from the fact that the set of feasible bad executions, as we will see, is regular as well, and that it can be recognised by an (effectively computable) non-deterministic finite state automaton. The decidability of k-synchronizability follows then from Lemma 22 and the decidability of the emptiness problem for non-deterministic finite state automata.

Recognition of feasible executions. We start with the automaton that recognises feasible executions; for this, we revisit the construction we just used for recognising sequences of k-exchanges that satisfy causal delivery.

In the remainder, we assume an execution $e' \in \mathit{asEx}(\mathfrak{S}')$ that contains exactly one send of the form $\mathit{send}(p, \pi, (q, \mathbf{v}))$ and one reception of the form $\mathit{rec}(\pi, q, \mathbf{v})$, this reception being the last action of e'. Let $(V, \{\xrightarrow{XY}\}_{X,Y \in \{R,S\}})$ be the conflict graph of e'. There are two uniquely determined vertices $\upsilon_{\mathsf{start}}, \upsilon_{\mathsf{stop}} \in V$ such that $\mathsf{proc}_R(\upsilon_{\mathsf{start}}) = \pi$ and $\mathsf{proc}_S(\upsilon_{\mathsf{stop}}) = \pi$ that correspond, respectively, to the first and last message exchanges of the deviation. The conflict graph of $e \cdot r$ is then obtained by merging these two nodes.

Lemma 23. *The execution e' is not feasible iff there is a vertex v in the conflict graph of e' such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v \xrightarrow{RR} \upsilon_{\mathsf{stop}}$.*

Proof. $\Leftarrow$ If there is v such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v \xrightarrow{RR} \upsilon_{\mathsf{stop}}$, this means that a message sent after the deviated message is received before it: hence, it violates causal delivery.

$\Rightarrow$ Assume now that $e \cdot r$ violates causal delivery and e does not. The only difference between the two is that an unmatched message becomes matched. It must therefore be the second item in Definition 4 that gets violated in $e \cdot r$, So the conflict graph $\mathsf{CG}(e \cdot r)$ contains two vertices v_d and v such that $v_d \overset{SS}{\dashrightarrow} v \xrightarrow{RR} v_d$, with $r \in v_d$. The node v_d becomes duplicated in $\upsilon_{\mathsf{start}}$ and υ_{stop} in $\mathsf{CG}(e')$, and therefore $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v \xrightarrow{RR} \upsilon_{\mathsf{stop}}$. □

In order to decide whether an execution e' is feasible, we want to forbid that a send action $\mathit{send}(p', q, \mathbf{v}')$ that happens causally after $\upsilon_{\mathsf{start}}$ is matched by a receive $\mathit{rec}(p', q, \mathbf{v}')$ that happens causally before the reception υ_{stop}. So we will consider sets of processes C_S^π and C_R^π similar to the ones used for $\overset{e,k}{\underset{\mathsf{cd}}{\Longrightarrow}}$,

$$\frac{\begin{array}{c}(\vec{l}, B) \xRightarrow[\mathsf{cd}]{e,k} (\vec{l'}, B') \qquad e = a_1 \cdots a_n \qquad (\forall v)\ \mathsf{proc}_S(v) \neq \pi \\ (\forall v, v')\ \mathsf{proc}_R(v) = \mathsf{proc}_R(v') = \pi \implies v = v' \wedge \mathtt{dest}_\pi = \bot \\ (\forall v)\ v \ni \mathit{send}(p, \pi, (q, \mathbf{v})) \implies \mathtt{dest}'_\pi = q \quad \mathtt{dest}_\pi \neq \bot \implies \mathtt{dest}'_\pi = \mathtt{dest}_\pi \\ {C^\pi_X}' = C^\pi_X \cup \{\mathsf{proc}_X(v') \mid v \overset{SS}{\dashrightarrow} v' \ \&\ v' \cap X \neq \emptyset \ \&\ (\mathsf{proc}_R(v) = \pi \text{ or } v = \psi_{\mathsf{start}})\} \\ \cup\{\mathsf{proc}_S(v) \mid \mathsf{proc}_R(v) = \pi \ \&\ X = S\} \\ \cup\{p \mid p \in C_{X,q} \ \&\ v \overset{SS}{\dashrightarrow} \psi_q \ \&\ (\mathsf{proc}_R(v) = \pi \text{ or } v = \psi_{\mathsf{start}})\} \\ \mathtt{dest}'_\pi \notin {C^\pi_R}'\end{array}}{(\vec{l}, B, C^\pi_S, C^\pi_R, \mathtt{dest}_\pi) \xRightarrow[\mathsf{feas}]{e,k} (\vec{l'}, B', {C^\pi_S}', {C^\pi_R}', \mathtt{dest}'_\pi)}$$

Figure 6: Definition of the relation $\xRightarrow[\mathsf{feas}]{e,k}$

but with the goal of computing which actions happen causally after the send to π. We also introduce a summary node ψ_{start} and the extra edges following the same principles as in the previous section. Formally, let $B : \mathbb{P} \to (2^{\mathbb{P}} \times 2^{\mathbb{P}})$, $C^\pi_S, C^\pi_R \subseteq \mathbb{P}$ and $e \in S^{\leq k}R^{\leq k}$ be fixed, and let $\mathsf{CG}(e, B) = (V', E')$ be the constraint graph with summary nodes for unmatched sent messages as defined in the previous section. The local constraint graph $\mathsf{CG}(e, B, C^\pi_S, C^\pi_R)$ is defined as the graph (V'', E'') where $V'' = V' \cup \{\psi_{\mathsf{start}}\}$ and E'' is E' augmented with

$$\begin{aligned} &\{\psi_{\mathsf{start}} \xrightarrow{SX} v \mid \mathsf{proc}_X(v) \in C^\pi_S \ \&\ v \cap X \neq \emptyset \text{ for some } X \in \{S, R\}\} \\ \cup\ &\{\psi_{\mathsf{start}} \xrightarrow{SS} v \mid \mathsf{proc}_X(v) \in C^\pi_R \ \&\ v \cap R \neq \emptyset \text{ for some } X \in \{S, R\}\} \\ \cup\ &\{\psi_{\mathsf{start}} \xrightarrow{SS} v \mid \mathsf{proc}_R(v) \in C^\pi_R \ \&\ v \text{ is unmatched}\} \ \cup\ \{\psi_{\mathsf{start}} \xrightarrow{SS} \psi_p \mid p \in C^\pi_R\} \end{aligned}$$

As before, we consider the "closure" $\overset{XY}{\dashrightarrow}$ of these edges by the rules of Figure 3. The transition relation $\xRightarrow[\mathsf{feas}]{e,k}$ is defined in Figure 6. It relates abstract configurations of the form $(\vec{l}, B, \vec{C}, \mathtt{dest}_\pi)$ with $\vec{C} = (C_{S,\pi}, C_{R,\pi})$ and $\mathtt{dest}_\pi \in \mathbb{P} \cup \{\bot\}$ storing to whom the message deviated to π was supposed to be delivered. Thus, the initial abstract configuration is $(l_0, B_0, (\emptyset, \emptyset), \bot)$, where $\bot$ means that the processus $\mathtt{dest}_\pi$ is undefined.

Lemma 24. *Let $e' = e'_1 \cdots e'_n \cdot \mathit{send}(\pi, q, \mathbf{v}) \cdot \mathit{rec}(\pi, q, \mathbf{v})$ with $e'_1, \ldots, e'_n \in S^{\leq k}R^{\leq k}$. Then e' is a k-synchronizable feasible execution of $\mathfrak{S}'$ iff there are $B' : \mathbb{P} \to 2^{\mathbb{P}}$, $\vec{C'} \in (2^{\mathbb{P}})^2$, and a tuple of control states $\vec{l'}$ such that $\pi \notin C_{R,q}$ (with $B'(q) = (C_{S,q}, C_{R,q})$), and*

$$(\vec{l_0}, B_0, (\emptyset, \emptyset), \bot) \xRightarrow[\mathsf{feas}]{e'_1,k} \ldots \xRightarrow[\mathsf{feas}]{e'_n,k} (\vec{l'}, B', \vec{C'}, q).$$

Proof. Let us first state what are the properties of the variables $\vec{C}, \mathtt{dest}_\pi$.

Let $e = e_1 \cdots e_n$ a k-synchronizable execution of $\mathfrak{S}'$ be fixed, and assume that there are $B, \vec{C}, \mathtt{dest}_\pi$ such that

$$(\vec{l_0}, B_0, \emptyset, \emptyset, \bot) \xRightarrow[\mathsf{feas}]{e_1,k} \ldots \xRightarrow[\mathsf{feas}]{e_n,k} (\vec{l}, B, C^\pi_S, C^\pi_R, \mathtt{dest}_\pi).$$

By induction on n, we want to establish that

1. $\mathtt{dest}_\pi = q$ if and only if a message of the form $(q, \mathbf{v})$ was sent to π in e;

2. there is at most one message sent to π in e;

3. let $\upsilon_{\mathsf{start}}$ denote the unique vertex in $\mathsf{CG}(e)$ (if it exists) such that $\mathsf{proc}_R(\upsilon_{\mathsf{start}}) = \pi$; for all $X \in \{S, R\}$,

$$C^\pi_X = \{\mathsf{proc}_X(v) \mid (v \cap X \neq \emptyset \;\&\; \upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v \text{ in } \mathsf{CG}(e)) \text{ or } (v, X) = (\upsilon_{\mathsf{start}}, S)\}.$$

The two first points easily follow from the definition of $\xRightarrow[\mathsf{feas}]{e,k}$. Let us focus on the last point. The case $n = 1$ is immediate. Let us assume that

$$(\vec{l_0}, B_0, (\emptyset, \emptyset), \bot) \xRightarrow[\mathsf{feas}]{e_1', k} \dots \xRightarrow[\mathsf{feas}]{e_{n-1}', k} (\vec{l}, B, \vec{C}, \mathtt{dest}_\pi) \xRightarrow[\mathsf{feas}]{e_n', k} (\vec{l'}, B', \vec{C'}, \mathtt{dest}'_\pi)$$

with $C^\pi_X = \{\mathsf{proc}_X(v) \mid (v \cap X \neq \emptyset \;\&\; \upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v \text{ in } \mathsf{CG}(e_1 \cdots e_{n-1})) \text{ or } (v, X) = (\upsilon_{\mathsf{start}}, S)\}$ and let us show that ${C^\pi_X}' = \{\mathsf{proc}_X(v) \mid (v \cap X \neq \emptyset \;\&\; \upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_n)$ or $(v, X) = (\upsilon_{\mathsf{start}}, S)\}$.

- Let $X \in \{S, R\}$ and $p \in {C^\pi_X}'$ and let us show that there is some v such that $p = \mathsf{proc}_X(v)$ and either $\upsilon_{\mathsf{start}} \overset{SX}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_n)$ or $(v, X) = (\upsilon_{\mathsf{start}}, S)$. We reason by case analysis on the reason why $p \in {C^\pi_X}'$, according to the definition of ${C^\pi_X}'$ in Figure 6.
  - $p \in C^\pi_X$. Then by induction hypothesis there is v such that $p = \mathsf{proc}_X(v)$, and $\upsilon_{\mathsf{start}} \xrightarrow{SS} v$ in $\mathsf{CG}(e_1 \cdots e_{n-1})$, and therefore also in $\mathsf{CG}(e_1 \cdots e_n)$, or $(v, X) = (\upsilon_{\mathsf{start}}, S)$.
  - $p = \mathsf{proc}_X(v')$, $v \overset{SS}{\dashrightarrow} v'$, $v' \cap X \neq \emptyset$, and $\mathsf{proc}_R(v) = \pi$, for some message exchanges v, v' of e_n. Since $\mathsf{proc}_R(v) = \pi$, $v = \upsilon_{\mathsf{start}}$. This shows this case.
  - $p = \mathsf{proc}_X(v')$, $\psi_{\mathsf{start}} \overset{SS}{\dashrightarrow} v'$, $v' \cap X \neq \emptyset$, for some message exchange v' of e_n. It remains to show that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v'$. From $\psi_{\mathsf{start}} \overset{SS}{\dashrightarrow} v'$, there are some v, Y such that $\psi_{\mathsf{start}} \xrightarrow{SY} v$ in $\mathsf{CG}(e_n, B, \vec{C})$, $v \cap Y \neq \emptyset$ and either $v \overset{YS}{\dashrightarrow} v'$ or $(v, Y) = (v', S)$. We reason by case analysis on the construction of the edge $\psi_{\mathsf{start}} \xrightarrow{SY} v$.
    - $*$ $\mathsf{proc}_Y(v) \in C^\pi_S$ and $v \cap Y \neq \emptyset$. Let $q = \mathsf{proc}_Y(v)$. Since $q \in C^\pi_S$, by induction hypothesis there is v_1 in a previous k-exchange such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1$ in $\mathsf{CG}(e_1 \cdots e_{n-1})$ or $v_1 = \upsilon_{\mathsf{start}}$. Since $\mathsf{proc}_S(v_1) = \mathsf{proc}_Y(v)$, there is an edge $v_1 \xrightarrow{SY} v$ in $\mathsf{CG}(e_1 \cdots e_n)$. By hypothesis, we also have either $v \overset{YS}{\dashrightarrow} v'$ or $(v, Y) = (v', S)$. So in both cases we get $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1 \xrightarrow{SS} v'$, or $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v'$ when $v_1 = \upsilon_{\mathsf{start}}$.
    - $*$ $\psi_{\mathsf{start}} \xrightarrow{SS} v$, $\mathsf{proc}_Y(v) \in C^\pi_R$ and $v \cap R \neq \emptyset$. Again by induction hypothesis, we have v_1 such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1 \xrightarrow{RY} v$, therefore $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v'$.

        - ∗ $\psi_{\mathsf{start}} \xrightarrow{SS} v$, $\mathsf{proc}_R(v) \in C^{\pi}_R$ and v unmatched. Again by induction hypothesis, we have v_1 such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1 \xrightarrow{RS} v$. If $v = v'$, we have $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v'$, which closes the case. Otherwise, from $v \overset{YS}{\dashrightarrow} v'$ and v unmatched we deduce $v \xrightarrow{SS} v'$; finally we $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1 \xrightarrow{RS} v \overset{SS}{\dashrightarrow} v'$, so $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v'$, which closes the case as well.
        - ∗ $v = \psi_q$ for some $q \in C^{\pi}_R$. Since ψ_q does not have outgoing edges of the form RS, $\psi_q \overset{SS}{\dashrightarrow} v'$. From $q \in C^{\pi}_R$, we get by induction hypothesis some node v_1 such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1$ and $\mathsf{proc}_R(v_1) = q$. As seen in the proof of Lemma 14, $\psi_q \overset{SS}{\dashrightarrow} v'$ implies that there is a vertex v_2 from a previous k-exchange that is an unmatched send to q such that $v_2 \overset{SS}{\dashrightarrow} v'$ in $\mathsf{CG}(e_1 \cdots e_n)$. Since v_1 is a matched send to q and v_2 is an unmatched send to q, by rule 4 in Figure 3 $v_1 \overset{SS}{\dashrightarrow} v_2$. All together, $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1 \overset{SS}{\dashrightarrow} v_2 \overset{SS}{\dashrightarrow} v'$, which closes this case.
    - – $p = \mathsf{proc}_X(v)$, $\mathsf{proc}_R(v) = \pi$, and $X = S$. Then $v = \upsilon_{\mathsf{start}}$, which closes this case.

- • Conversely, let us show that for all $X \in \{S, R\}$ and v such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_n)$, $\mathsf{proc}_S(v) \neq \pi$, and $v \cap X \neq \emptyset$, it holds that $\mathsf{proc}_X(v) \in {C^{\pi}_X}'$ (the corner case to be proved, $(v, X) = (\upsilon_{\mathsf{start}}, S)$, is treated in the last item). Again, we reason by induction on the number n of k-exchanges. If $n = 0$, it is immediate as there are no such v, X. Let us assume that the property holds for all choices of v_1, X_1 such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1$ in $\mathsf{CG}(e_1 \cdots e_{n-1})$, $\mathsf{proc}_S(v_1) \neq \pi$, and $v_1 \cap X_1 \neq \emptyset$. Let v, X be fixed with $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_n)$, and $v \cap X \neq \emptyset$, and let us show that $\mathsf{proc}_X(v) \in {C^{\pi}_X}'$. We reason by case analysis on the occurrence in e_n, or not, of both $\upsilon_{\mathsf{start}}$ and v.

    - – $\upsilon_{\mathsf{start}}$ and v are in e_n. Then from $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_n)$ and the proof of Lemma 14, we get that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e_n, B)$. By definition of ${C^{\pi}_S}'$ (first line), it contains $\mathsf{proc}_X(v)$
    - – $\upsilon_{\mathsf{start}}$ in e_n and v in $e_1 \cdots e_{n-1}$. Then there are v_1, v_2, q such that
        - ∗ v_1 is in e_n, and either $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1$ in $\mathsf{CG}(e_1 \cdots e_n)$ or $v_1 = \upsilon_{\mathsf{start}}$,
        - ∗ v_2 is in $e_1 \cdots e_{n-1}$, $v_1 \overset{SS}{\dashrightarrow} v_2$ by rule 4 of Figure 3, i.e., v_1 is a matched send to q and v_2 is an unmatched send to q
        - ∗ either $v_2 \overset{X_2 S}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_{n-1})$, or $v_2 = v$

      From the first item, by the proof of Lemma 14, we get either $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v_1$ in $\mathsf{CG}(e_n, B)$ or $v = v_1$. From the second item, we get $v_1 \xrightarrow{SS} \psi_q$ in $\mathsf{CG}(e_n, B)$. From these two, we get $\psi_{\mathsf{start}} \overset{SS}{\dashrightarrow} \psi_p$ in $\mathsf{CG}(e_n, B, \vec{C})$. By definition of C^{π}_X, we therefore have $C_{X,q} \subseteq C^{\pi}_X$. From the third item, we get $\mathsf{proc}_X(v) \in C_{X,q}$. So finally $\mathsf{proc}_X(v) \in C^{\pi}_X$.
    - – $\upsilon_{\mathsf{start}}$ in $e_1 \cdots e_{n-1}$ and v in e_n. Then there are v_1, v_2, Y, Z such that

    - ∗ either $v_{\mathsf{start}} \overset{SY}{\dashrightarrow} v_1$ in $\mathsf{CG}(e_1 \cdots e_{n-1})$, or $(v, S) = (v_1, Y)$
      - ∗ $v_1 \xrightarrow{YZ} v_2$
      - ∗ either $v_2 \overset{ZS}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_n)$, with both v_2 and v in e_n, or $(v_2, Z) = (v, S)$

    From the first item, by induction hypothesis, we get $\mathsf{proc}_Y(v_1) \in C^{\pi}_{X}$. From the second item, we get $\mathsf{proc}_Y(v_1) = \mathsf{proc}_Z(v_2)$, and from the definition of outgoing edges of ψ_{start}, we get $\psi_{\mathsf{start}} \xrightarrow{SZ} v_2$ in $\mathsf{CG}(e_n, B, \vec{C})$. From the third item and the proof of Lemma 14, we get either $v_2 \overset{ZS}{\dashrightarrow} v$ in $\mathsf{CG}(e_n, B)$ or $(v_2, Z) = (v, S)$. All together, we get $\psi_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e_n, B, \vec{C})$. By definition of $C^{\pi}_{S}{}'$ (first line), it contains $\mathsf{proc}_X(v)$.

  - v_{start} and v in $e_1 \cdots e_{n-1}$. If $v_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_{n-1})$, then $\mathsf{proc}_X(v) \in C^{\pi}_{R}$ holds immediately by induction hypothesis. Otherwise, there are $v_1, v_2, v_3, v_4, Y, Z, q$ such that
    - ∗ either $v_{\mathsf{start}} \overset{SY}{\dashrightarrow} v_1$ in $\mathsf{CG}(e_1 \cdots e_{n-1})$, or $(v, S) = (v_1, Y)$
    - ∗ $v_1 \xrightarrow{YZ} v_2$
    - ∗ either $v_2 \overset{ZS}{\dashrightarrow} v_3$ in $\mathsf{CG}(e_1 \cdots e_n)$, with both v_2 and v_3 in e_n, or $(v_2, Z) = (v_3, S)$
    - ∗ $v_3 \overset{SS}{\dashrightarrow} v_4$ due to rule 4 in Figure 3, i.e., v_3 is a matched send to q and v_4 is an unmatched send to q
    - ∗ either $v_4 \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e_1 \cdots e_{n-1})$, or $(v_4, T) = (v, S)$

    From the first item, by induction hypothesis, we get $\mathsf{proc}_Y(v_1) \in C^{\pi}_{X}$. From the second item, we get $\mathsf{proc}_Y(v_1) = \mathsf{proc}_Z(v_2)$, and from the definition of outgoing edges of ψ_{start}, we get $\psi_{\mathsf{start}} \xrightarrow{SZ} v_2$ in $\mathsf{CG}(e_n, B, \vec{C})$. From the third item and the proof of Lemma 14, we get either $v_2 \overset{ZS}{\dashrightarrow} v_3$ in $\mathsf{CG}(e_n, B)$ or $(v_2, Z) = (v_3, S)$. From the fourth item, we get $v_3 \overset{SS}{\dashrightarrow} \psi_q$ in $\mathsf{CG}(e_n, B)$. To sum up, we have $\psi_{\mathsf{start}} \xrightarrow{SS} \psi_q$ in $\mathsf{CG}(e_n, B, \vec{C})$. By definition of C^{π}_{X}, we therefore have $C_{X,q} \subseteq C^{\pi}_{X}$. From the fifth item, we get by the proof of Lemma 14 that $\mathsf{proc}_X(v) \in C_{X,q}$, which ends this case.

- Finally, let us finish the proof of the converse implication, and show the remaining case, i.e., let us show that $\mathsf{proc}_S(v_{\mathsf{start}}) \in C^{\pi}_{S}$. This is immediate from the definition of $C^{\pi}_{S}{}'$ (cfr. the set $\{\mathsf{proc}_S(v) \mid \mathsf{proc}_R(v) = \pi \;\&\; X = S\}$).

We are done with proving that $C^{\pi}_{X} = \{\mathsf{proc}_X(v) \mid (v \cap X \neq \emptyset \;\&\; v_{\mathsf{start}} \overset{SS}{\dashrightarrow} v \text{ in } \mathsf{CG}(e)) \text{ or } (v, X) = (v_{\mathsf{start}}, S)\}$. It is now time to conclude with the proof of Lemma 24 itself.

Let $e' = e'_1 \cdots e'_n \cdot \mathit{send}(\pi, q, \mathbf{v}) \cdot \mathit{rec}(\pi, q, \mathbf{v})$ with $e'_1, \cdots e'_n \in S^{\leq k} R^{\leq k}$ be fixed.

$\Leftarrow$ Let us assume that e' is a k-synchronizable feasible execution of $\mathfrak{S}'$ and let us show that

$$(\vec{l_0}, B_0, (\emptyset, \emptyset), \bot) \xRightarrow[\mathsf{feas}]{e'_1, k} \ldots \xRightarrow[\mathsf{feas}]{e'_n, k} (\vec{l'}, B', \vec{C'}, \mathtt{dest}_{\pi}).$$

for some $B', \vec{C'}, \mathtt{dest}_\pi$ with $\pi \notin C_{R,\mathtt{dest}_\pi}$. By definition of $\xRightarrow[\mathsf{feas}]{e,k}$, B', $\vec{C'}$ and $\mathtt{dest}_\pi$ are uniquely determined, and it is enough to prove that $\mathtt{dest}_\pi \notin C^\pi_R$. Let us assume by absurd that $\mathtt{dest}_\pi \in C^\pi_R$. Then, by the property we just proved, there is v such that $\mathsf{proc}_R(v) = \mathtt{dest}_\pi$, $v \cap R \neq \emptyset$, and $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e'_1 \cdots e'_n)$. So we get $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v \xrightarrow{RR} \upsilon_{\mathsf{stop}}$ in $\mathsf{CG}(e')$, and by Lemma 23, e' should not be feasible: contradiction. Finally, $\pi \notin C_{R,\pi}$ because e', as an execution of $\mathfrak{S}'$, satisfies causal delivery.

$\Rightarrow$ Let us assume that

$$(\vec{l_0}, B_0, (\emptyset, \emptyset), \bot) \xRightarrow[\mathsf{feas}]{e'_1,k} \dots \xRightarrow[\mathsf{feas}]{e'_n,k} (\vec{l'}, B', \vec{C'}, \mathtt{dest}_\pi).$$

for some $B', \vec{C'}, \mathtt{dest}_\pi$ with $\pi \notin C_{R,\mathtt{dest}_\pi}$, and let us show that e' is a k-synchronizable feasible execution of $\mathfrak{S}'$. From the definition of $\xRightarrow[\mathsf{feas}]{e,k}$, we get

$$(\vec{l_0}, B_0) \xRightarrow[\mathsf{cd}]{e'_1,k} \dots \xRightarrow[\mathsf{cd}]{e'_n,k} (\vec{l'}, B')$$

and from Lemma 14, $e'_1 \cdots e'_n$ is k-synchronizable. Since the last two actions $send(\pi, q, \mathbf{v}) \cdot rec(\pi, q, \mathbf{v})$ can be placed in a specific k-exchange, and since they do not break causal delivery (because $\pi \notin C_{R,\mathtt{dest}_\pi})$), e' is a k-synchronizable execution of $\mathfrak{S}'$. It remains to show that e' is feasible. Again, let us reason by contradiction and assume that e' is not feasible. By Lemma 23, there is v such that $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v \xrightarrow{RR} \upsilon_{\mathsf{stop}}$ in $\mathsf{CG}(e')$. In other words, $\mathsf{proc}_R(v) = \mathtt{dest}_\pi$, $v \cap R \neq \emptyset$, and $\upsilon_{\mathsf{start}} \overset{SS}{\dashrightarrow} v$ in $\mathsf{CG}(e'_1 \cdots e'_n)$. So, by the property we just proved, $\mathtt{dest}_\pi \in C^\pi_R$, and the contradiction. □

Recognition of bad executions. Finally, we define a non-deterministic finite state automaton that recognizes bad executions, i.e., feasible executions $e' = \mathsf{deviate}(e \cdot r)$ such that $e \cdot r$ is not k-synchronous. We come back to the "non-extended" conflict graph, without edges of the form $\overset{XY}{\dashrightarrow}$. Let $\mathsf{Post}^*(v) = \{v' \in V \mid v \rightarrow^* v'\}$ be the set of vertices reachable from v (not necessarily through a causal path), and let $\mathsf{Pre}^*(v) = \{v' \in V \mid v' \rightarrow^* v\}$ be the set of vertices co-reachable from v. For a set of vertices $U \subseteq V$, let $\mathsf{Post}^*(U) = \bigcup\{\mathsf{Post}^*(v) \mid v \in U\}$, and $\mathsf{Pre}^*(U) = \bigcup\{\mathsf{Pre}^*(v) \mid v \in U\}$.

Lemma 25. *The feasible execution e' is bad iff one of the two holds*

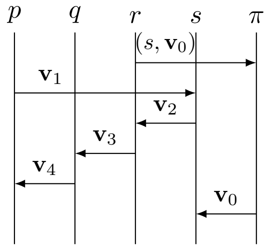
i) $\upsilon_{\mathsf{start}} \longrightarrow^* \xrightarrow{RS} \longrightarrow^* \upsilon_{\mathsf{stop}}$, or

ii) the size of the set $\mathsf{Post}^*(\upsilon_{\mathsf{start}}) \cap \mathsf{Pre}^*(\upsilon_{\mathsf{stop}})$ is greater or equal to $k+2$.

Proof. Since e' is k-synchronous, e (without the last reception r) is k-synchronous. By Theorem 8, e' is bad if and only if $\mathsf{CG}(e \cdot r)$ contains either a cyclic path with an RS edge, or a SCC with of size $\geq k+1$. This cyclic path (resp. SCC) must contain the vertex associated with the last receive r of $e \cdot r$. In $\mathsf{CG}(e')$, this cyclic (resp. SCC) corresponds to a path from $\upsilon_{\mathsf{start}}$ to υ_{stop} (resp. the set of vertices that are both reachable from $\upsilon_{\mathsf{start}}$ and co-reachable from υ_{stop}). Since the $\upsilon_{\mathsf{start}}$ and υ_{stop} account for the same node in the conflict graph of $e \cdot r$, the size of the SCC is one less than the size of the set $\mathsf{Post}^*(\upsilon_{\mathsf{start}}) \cap \mathsf{Pre}^*(\upsilon_{\mathsf{stop}})$. □

In order to determine whether a given message exchange v of $\mathsf{CG}(e')$ should be counted as reachable (resp. co-reachable), we will compute at the entry and exit of every k-exchange of e' which processes are "reachable" or "co-reachable".

Example 26. *(Reachable and co-reachable processes)*



$msc(e)$

Consider the MSC of an execution e on the right composed of five 1-exchanges. While sending message $(s, \mathbf{v}_0)$ that corresponds to $\upsilon_{\mathsf{start}}$, process r becomes "reachable": any subsequent message exchange that involves r corresponds to a vertex of the conflict graph that is reachable from $\upsilon_{\mathsf{start}}$. While sending $\mathbf{v}_2$, process s becomes "reachable", because process r will be reachable when it will receive message $\mathbf{v}_2$. Similary, q becomes reachable after receiving $\mathbf{v}_3$ because r was reachable when it sent $\mathbf{v}_3$, and p becomes reachable after receiving $\mathbf{v}_4$ because q was reachable when it sent it. Co-reachability works similarly, but reasoning backwards on the timelines. For instance, process s stops being "co-reachable" while it receives $\mathbf{v}_0$, process r stops being co-reachable after it receives $\mathbf{v}_2$, and process p stops being co-reachable by sending $\mathbf{v}_1$. The only message that is sent by a process being both reachable and co-reachable at the instant of the sending is $\mathbf{v}_2$, therefore it is the only message that will be counted as contributing to the SCC.

More formally, let e be an execution, $\mathsf{CG}(e)$ its conflict graph and P, Q two sets of processes, $\mathsf{Post}_e(P) = \mathsf{Post}^*\Big(\{v \mid \mathsf{procs}(v) \cap P \neq \emptyset\}\Big)$ and $\mathsf{Pre}_e(Q) = \mathsf{Pre}^*\Big(\{v \mid \mathsf{procs}(v) \cap Q \neq \emptyset\}\Big)$ are introduced to represent the local view through k-exchanges of $\mathsf{Post}^*(\upsilon_{\mathsf{start}})$ and $\mathsf{Pre}^*(\upsilon_{\mathsf{stop}})$. For instance, for e as in Example 26, we get $\mathsf{Post}_e(\{\pi\}) = \{(s, \mathbf{v}_0), \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_0\}$ and $\mathsf{Pre}_e(\{\pi\}) = \{\mathbf{v}_0, \mathbf{v}_2, \mathbf{v}_1, (s, \mathbf{v}_0)\}$. In each k-exchange e_i the size of the intersection between $\mathsf{Post}_{e_i}(P)$ and $\mathsf{Pre}_{e_i}(Q)$ will give the local contribution of the current k-exchange to the calculation of the size of the global SCC. In the transition relation $\xRightarrow[\mathsf{bad}]{e,k}$ this value is stored in variable $\texttt{cnt}$. The last ingredient to consider is to recognise if an edge RS belongs to the SCC. To this aim, we use a function $\texttt{lastisRec} : \mathbb{P} \to \{\mathsf{True}, \mathsf{False}\}$ that for each process stores the information whether the last action in the previous k-exchange was a reception or not. Then depending on the value of this variable and if a node is in the current SCC or not the value of $\texttt{sawRS}$ is set accordingly.

The transition relation $\xRightarrow[\mathsf{bad}]{e,k}$ defined in Figure 7 deals with abstract configurations of the form $(P, Q, \texttt{cnt}, \texttt{sawRS}, \texttt{lastisRec}')$ where $P, Q \subseteq \mathbb{P}$, $\texttt{sawRS}$ is a Boolean value, and $\texttt{cnt}$ is a counter bounded by $k+2$. We denote by $\texttt{lastisRec}_0$ the function where all $\texttt{lastisRec}(p) = \mathsf{False}$ for all $p \in \mathbb{P}$.

Lemma 27. *Let $e' = e'_1 \cdots e'_n$ be a k-synchronizable feasible execution of $\mathfrak{S}'$. Then e' is bad iff there are $P', Q \subseteq \mathbb{P}$, $\mathsf{sawRS} \in \{\mathsf{True}, \mathsf{False}\}$, $\mathsf{cnt} \in \{0, \ldots, k+2\}$, such that*

$$(\{\pi\}, Q, 0, \mathsf{False}, \texttt{lastisRec}_0) \xRightarrow[\mathsf{bad}]{e'_1,k} \ldots \xRightarrow[\mathsf{bad}]{e'_n,k} (P', \{\pi\}, \texttt{cnt}, \texttt{sawRS}, \texttt{lastisRec})$$

$$\frac{\begin{array}{c} P' = \mathsf{procs}(\mathsf{Post}_e(P)) \qquad Q = \mathsf{procs}(\mathsf{Pre}_e(Q')) \qquad SCC_e = \mathsf{Post}_e(P) \cap \mathsf{Pre}_e(Q') \\ \mathtt{cnt}' = \mathsf{min}(k+2, \mathtt{cnt} + n) \quad \text{where } n = |SCC_e| \\ \mathtt{lastisRec}'(q) \Leftrightarrow (\exists v \in SCC_e.\mathsf{proc}_R(v) = q \wedge v \cap R \neq \emptyset) \vee (\mathtt{lastisRec}(q) \wedge \nexists v \in V.\mathsf{proc}_S(v) = q) \\ \mathtt{sawRS}' = \mathtt{sawRS} \vee (\exists v \in SCC_e)(\exists p \in \mathbb{P} \setminus \{\pi\})\ \mathsf{proc}_S(v) = p \wedge \mathtt{lastisRec}(p) \wedge p \in P \cap Q \end{array}}{(P, Q, \mathtt{cnt}, \mathtt{sawRS}, \mathtt{lastisRec}) \xRightarrow[\mathsf{bad}]{e,k} (P', Q', \mathtt{cnt}', \mathtt{sawRS}', \mathtt{lastisRec}')}$$

Figure 7: Definition of the relation $\xRightarrow[\mathsf{bad}]{e,k}$

and at least one of the two holds: either $\mathtt{sawRS} = \mathsf{True}$*, or* $\mathtt{cnt} = k + 2$*.*

Proof. $\Rightarrow$ Let us suppose $e' = e'_1 \cdots e'_n$ be a k-synchronous bad and feasible execution. We show that

$$(\{\pi\}, Q, \mathsf{False}, 0) \xRightarrow[\mathsf{bad}]{e'_1,k} \ldots \xRightarrow[\mathsf{bad}]{e'_n,k} (P', \{\pi\}, \mathtt{sawRS}, \mathtt{cnt})$$

for some P', Q and either $\mathtt{sawRS} = \mathsf{True}$, or $\mathtt{cnt} = k + 2$.

We proceed by induction on n.

Base n=2 Notice that, for a feasible execution, there are at least two k-exchanges as the deviation cannot fit a single k-exchange: the send from process π to the original recipient must follow the reception of the deviated message, thus it has to belong to a subsequent k-exchange. Then $e' = e'_1 \cdot e'_2$ and we show $(\{\pi\}, Q, \mathsf{False}, 0) \xRightarrow[\mathsf{bad}]{e'_1,k} (P', Q', \mathtt{sawRS}', \mathtt{cnt}) \xRightarrow[\mathsf{bad}]{e'_2,k} (P'', \{\pi\}, \mathtt{sawRS}, \mathtt{cnt}')$.

By Lemma 25, we have that either $v_{\mathsf{start}} \longrightarrow^* \xrightarrow{RS} \longrightarrow^* v_{\mathsf{stop}}$, or the size of the set $\mathsf{Post}^*(v_{\mathsf{start}}) \cap \mathsf{Pre}^*(v_{\mathsf{stop}})$ is greater or equal to $k + 2$.

If $v_{\mathsf{start}} \longrightarrow^* \xrightarrow{RS} \longrightarrow^* v_{\mathsf{stop}}$, then since a label RS cannot exists in a local conflict graph, there exist two paths $v_{\mathsf{start}} \longrightarrow^* v_1$ in $\mathsf{CG}(e_1)$ and $v_2 \longrightarrow^* v_{\mathsf{stop}}$ in $\mathsf{CG}(e_2)$, with $\mathsf{proc}_R(v_1) = \mathsf{proc}_S(v_2)$. We have that $v_2 \in \mathsf{Pre}_{e_2}(\pi)$ and $\mathtt{lastisRec}(\mathsf{proc}_S(v_2))$ is True, thus $\mathtt{sawRS}$ becomes True concluding this part of the proof.

Now suppose that the size of the set $\mathsf{Post}^*(v_{\mathsf{start}}) \cap \mathsf{Pre}^*(v_{\mathsf{stop}})$ is greater or equal to $k + 2$. We show that all nodes in $\mathsf{Post}^*(v_{\mathsf{start}}) \cap \mathsf{Pre}^*(v_{\mathsf{stop}})$ have been counted either in the first or in the second k-exchange. Take $v \in \mathsf{Post}^*(v_{\mathsf{start}}) \cap \mathsf{Pre}^*(v_{\mathsf{stop}})$ then there exists a path $v_{\mathsf{start}} \longrightarrow^* v \longrightarrow^* v_{\mathsf{stop}}$ and v is an exchange that belongs either to the first or the second k-exchange. If v belongs to the first one then we can divide previous path in two parts such that $v_{\mathsf{start}} \longrightarrow^* v \longrightarrow^* v_1$ is in $\mathsf{CG}(e_1)$, $v_2 \longrightarrow^* v_{\mathsf{stop}}$ is in $\mathsf{CG}(e_2)$ and $\mathsf{procs}(v_1) \cup \mathsf{procs}(v_2) = \{p\} \neq \emptyset$. From this it follows that process $p \in Q'$ and thus $v \in \mathsf{Pre}_{e_1}(Q')$. Moreover, $v \in \mathsf{Post}_{e_1}(\pi)$ and therefore the node v is counted in the first k-exchange.

Similarly, if v belongs to the second k-exchange, we can divide the path into two parts such that $v_{\mathsf{start}} \longrightarrow^* v_1$ is in $\mathsf{CG}(e_1)$, $v_2 \longrightarrow^* v \longrightarrow^* v_1 v_{\mathsf{stop}}$ is in $\mathsf{CG}(e_2)$ and $\mathsf{procs}(v_1) \cup \mathsf{procs}(v_2) = \{p\} \neq \emptyset$. From this it follows that process $p \in P'$ and thus $v \in \mathsf{Post}_{e_2}(P')$. Moreover, $v \in \mathsf{Pre}_{e_2}(\pi)$ and therefore the node v is counted in the second k-exchange.

Thus all nodes in $\mathsf{Post}^*(\upsilon_{\mathsf{start}}) \cap \mathsf{Pre}^*(\upsilon_{\mathsf{stop}})$ are considered and if $\mathsf{Post}^*(\upsilon_{\mathsf{start}}) \cap \mathsf{Pre}^*(\upsilon_{\mathsf{stop}}) \geq k+2$ so is variable $\mathtt{cnt}'$, concluding this part of the proof.

Inductive step It is an easy generalisation of what has been said in the previous part of the proof. By considering that by inductive hypothesis sets $\mathsf{Post}_{e_i}(P)$ and $\mathsf{Pre}_{e_i}(Q)$ contains respectively all the processes that are reachable from the exchange to process π and are co-reachable from the exchange from process π.

$\Leftarrow$ Let $e' = e'_1 \cdots e'_n$ a k-synchronous feasible execution and P', $Q \subseteq \mathbb{P}$, $\mathtt{sawRS} \in \{\mathsf{True}, \mathsf{False}\}$, $\mathtt{cnt} \in \{0, \ldots, k+2\}$ such that

$$(\{\pi\}, Q, \mathsf{False}, 0) \xRightarrow[\mathsf{bad}]{e'_1,k} \ldots \xRightarrow[\mathsf{bad}]{e'_n,k} (P', \{\pi\}, \mathtt{sawRS}, \mathtt{cnt})$$

We have either $\mathtt{sawRS} = \mathsf{True}$ or $\mathtt{cnt} = k+2$.

1. We suppose that $\mathtt{sawRS} = \mathsf{True}$. If $\mathtt{sawRS} = \mathsf{True}$ then $\exists e'_i$ where $\mathtt{sawRS} = \mathsf{False}$ and $\mathtt{sawRS}' = \mathsf{True}$. In this k-exchange, $\exists p \in \mathbb{P}$ such that $p \in P$, $\mathtt{lastisRec}(p) = \mathsf{True}$ and $\exists v$ such that $\mathsf{proc}_S(v) = p$ and $v \in \mathsf{Pre}_e(Q')$. Since $p \in P$, then there is a path $\upsilon_{\mathsf{start}} \longrightarrow^* \xrightarrow{RS} v$ in $\mathsf{CG}(e)$. On the other hand, since $v \in \mathsf{Pre}_e(Q')$ then $v \in \mathsf{Pre}^*(\upsilon_{\mathsf{stop}})$ and there is a path $v \longrightarrow^* \upsilon_{\mathsf{stop}}$ in $\mathsf{CG}(e)$. Therefore, there is a path $\upsilon_{\mathsf{start}} \longrightarrow^* \xrightarrow{RS} v \longrightarrow^* \upsilon_{\mathsf{stop}}$ in $\mathsf{CG}(e)$ and so e' is bad.

2. We suppose that $\mathtt{cnt} = k+2$. As previously, e' is feasible by Lemma 24. Each v belongs to $\mathsf{Post}_{e'_i}(P_i) \cap \mathsf{Pre}_{e'_i}(Q'_i) \backslash \upsilon_{\mathsf{start}}$ also belongs to $\mathsf{Post}^*(\upsilon_{\mathsf{start}}) \cap \mathsf{Pre}^*(\upsilon_{\mathsf{stop}})$ then $\mid \mathsf{Post}^*(\upsilon_{\mathsf{start}}) \cap \mathsf{Pre}^*(\upsilon_{\mathsf{stop}}) \mid \geq k+2$. Therefore, e' is bad.

Therefore, in both cases, e' is feasible and bad, concluding the proof.

□

We can finally conclude this section proving the decidability of k-synchronizability.

Theorem 28. *The k-synchronizability of a system $\mathfrak{S}$ is decidable for $k \geq 1$.*

Proof. Let $\mathfrak{S}$ be fixed. By Lemmata 22, 24, and 27, $\mathfrak{S}$ is not k-synchronizable if and only if there is a sequence of actions $e' = e'_1 \cdots e'_n \cdot s \cdot r$ such that $e_i \in S^{\leq k} R^{\leq k}$, $s = send(\pi, q, \mathbf{v})$, $r = rec(\pi, q, \mathbf{v})$,

$$(\vec{l_0}, B_0, (\emptyset, \emptyset), \bot) \xRightarrow[\mathsf{feas}]{e'_1,k} \ldots \xRightarrow[\mathsf{feas}]{e'_n,k} (\vec{l'}, B', \vec{C'}, q)$$

and

$$(\{\pi\}, Q, \mathsf{False}, 0) \xRightarrow[\mathsf{bad}]{e'_1,k} \ldots \xRightarrow[\mathsf{bad}]{e'_n,k} \xRightarrow[\mathsf{bad}]{s \cdot r,k} (P', \{\pi\}, \mathtt{sawRS}, \mathtt{cnt})$$

for some $\vec{l'}, B', \vec{C'}, Q, P'$ with $\pi \notin C_{R,q}$. Since both relations $\xRightarrow[\mathsf{feas}]{e,k}$ and $\xRightarrow[\mathsf{bad}]{e,k}$ are finite state, the existence of such a sequence of actions is decidable. □

6 Comparison with [4]

We just showed that whether a system is k-synchronizable for a fixed k is decidable, and we also showed that the reachability problem for k-synchronizable systems is decidable. Our proof may seem, on the surface, quite similar to the one presented in [4], and it could be believed that we only corrected a few minor typos. This section intends to expose some of the flaws we found in [4] so as to defend that, on the contrary, the changes we introduced were far from foreseeable in the original presentation.

Differences in the graphical characterisation of k-synchronizable executions

In [4], Bouajjani *et al.* give the following characterisation of k-synchronous executions in terms of a structural property of the conflict graphs of their MSC.

> **Characterisation of k-synchronizability [4, Theorem 1].** A MSC t[1] satisfying causal delivery is k-synchronous iff every *cycle* in its conflict graph does not contain a *RS* edge and is of size at most k.

If the word *cycle* in this statement means *Hamiltonian cycle* (i.e., a cyclic path that does not go twice through the same vertex), then this statement is not correct. Indeed, consider again Example 9. This graph is not Hamiltonian, and the largest Hamiltonian cycle indeed is of size 4 only. But as we already discussed in Example 9, the corresponding MSC is not 4-synchronous.

It is true that the word cycle could be understood, with some open mindedness, as equivalent to SCC. But the subsequent developments in [4] indicate that what is meant by *cycle* in the above statement really is Hamiltonian cycle. In particular, the algorithm that is later used for deciding whether a system is k-synchronizable is not correct either: the MSC of Figure 2c would be considered as 4-synchronizable according to this algorithm.

Differences in the definition of $\overset{e,k}{\underset{\mathsf{cd}}{\Longrightarrow}}$

In [4] the authors define $\overset{e,k}{\underset{\mathsf{cd}}{\Longrightarrow}}$ in a rather different way: they do not explicitly give a graphical characterisation of causal delivery; instead they compute, for every process p, the set $B(p)$ of processes that either sent an unmatched message to p or received a message from a process in $B(p)$. They then make sure that any message sent to p by a process $q \in B(p)$ is unmatched. According to that, relation $\overset{e,k}{\underset{\mathsf{cd}}{\Longrightarrow}}$ in [4], considers that the following execution (see also Example 11)

$$\begin{aligned} &\mathit{send}(q, r, \mathbf{v}_1) \cdot \mathit{send}(q, s, \mathbf{v}_2) \cdot \mathit{rec}(q, s, \mathbf{v}_2) \cdot \mathit{send}(p, s, \mathbf{v}_3) \cdot \mathit{rec}(p, s, \mathbf{v}_3) \cdot \\ &\qquad\qquad\qquad\qquad\qquad\qquad \mathit{send}(p, r, \mathbf{v}_4) \cdot \mathit{rec}(p, r, \mathbf{v}_4) \end{aligned}$$

satisfies causal delivery and is 1-synchronous. However, this is not the case: we saw in Example 11 that neither this execution nor any causally equivalent satisfy causal delivery.

[1] A trace t, as in the original terminology.

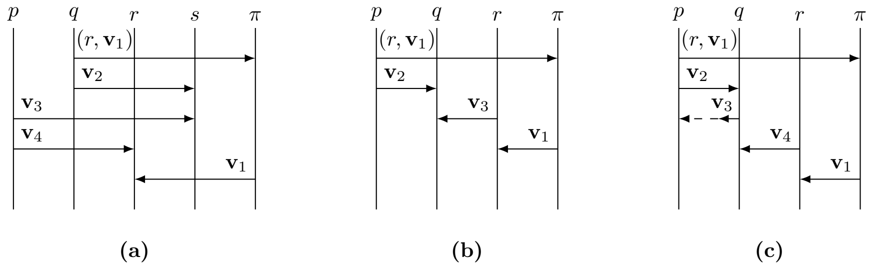


Figure 8: MSCs of problematic executions

Differences in the definition of $\xRightarrow[\mathsf{feas}]{e,k}$

In [4] the authors verify that a trace is feasible with a *monitor* which reviews the actions of the execution and adds processes that no longer are allowed to send a message to the receiver of π, in the same way as function B. According to this monitor, the following execution $e' = \mathsf{deviate}(e \cdot r)$ (MSC in Figure 8a) is feasible, i.e., is runnable in $\mathfrak{S}'$ and $e \cdot r$ is runnable in $\mathfrak{S}$.

$$e' = send(q, \pi, (r, \mathbf{v}_1)) \cdot rec(q, \pi, (r, \mathbf{v}_1)) \cdot send(q, s, \mathbf{v}_2) \cdot rec(q, s, \mathbf{v}_2) \cdot send(p, s, \mathbf{v}_3) \cdot rec(p, s, \mathbf{v}_3) \cdot send(p, r, \mathbf{v}_4) \cdot rec(p, r, \mathbf{v}_4) send(\pi, r, \mathbf{v}_1) \cdot rec(\pi, r, \mathbf{v}_4)$$

However, as previously, this execution is not feasible because there is a causal dependency between $\mathbf{v}_1$ and $\mathbf{v}_3$. This execution would then be considered as feasible and therefore would belong to set $sTr_k(\mathfrak{S}')$. As there is no corresponding execution in $asTr(\mathfrak{S})$, the comparison and therefore the k-synchronizability, could be distorted and appear as a false negative.

Differences in the definition of $\xRightarrow[\mathsf{bad}]{e,k}$

As for the notion of feasibility, to determine if an execution is bad, in [4] the authors use a monitor that builds a path between the send to process π and the send from π. In addition to the problems related to the wrong characterisation of k-synchronizability, this monitor not only can detect an RS edge when there should be none, but also it can miss them when they exist. In general, the problem arises because the path is constructed by considering only an endpoint at the time. Figure 8b depicts the MSC associated to an execution feasible and without label RS. Still, the monitor considers the reception of $\mathbf{v}_2$ followed by the send of $\mathbf{v}_3$. A label RS is thus wrongly detected. Conversely, Figure 8c depicts the MSC associated to an execution feasible but bad. With the monitor in [4], the action seen after the send of $\mathbf{v}_3$ is the send of $\mathbf{v}_4$ and so the existing label RS is ignored at the profit of a non existing label SS.

Other differences

Due to to the above errors, that concern fundamental points in [4], we had to propose a considerably different approach. The extended edges of the conflict graph, and the graphical characterisation of causal delivery as well as summary nodes, have no equivalent in [4]. As underlined above, transition relations $\xRightarrow[\mathsf{feas}]{e,k}$ and $\xRightarrow[\mathsf{bad}]{e,k}$ build on the graphical characterisations of causal delivery and k-synchronizability, they depart considerably from the proposal in [4].

7 Concluding remarks and related works

In this paper we have studied k-synchronizability for systems communicating via mailboxes. We have corrected the reachability and decidability proofs introduced in [4] by proposing a more involved characterisation via conflict graphs of k-synchronizability.

We conclude by commenting on some related works. The idea of "communication layers" is present in the early works of Elrad and Francez [8] or Chou and Gafni [7]. More recently, Chaouch-Saad *et al.* [6] verified some consensus algorithms using the Heard-Of Model that proceeds by "communication-closed rounds".

The concept that an asynchronous system may have an "equivalent" synchronous counterpart has also been widely studied. Lipton's reduction [13] reschedules an execution so as to move the receive actions as close as possible from their corresponding send. Reduction recently received an increasing interest for verification purpose, e.g. by Kragl *et al.* [11], or Gleissenthal *et al.* [15].

Existentially bounded communication systems have been studied by Genest *et al.* [10, 14]: a system is existentially k-bounded if any execution can be rescheduled in order to become k-bounded. This approach targets a broader class of systems than k-synchronizability, because it does not require that the execution can be chopped in communication-closed rounds. In the perspective of the current work, an interesting result is the decidability of existential k-boundedness for deadlock-free systems of communicating machines with peer-to-peer channels. Despite the more general definition, these older results are incomparable with the present ones, that deal with systems communicating with mailboxes, and not peer-to-peer channels.

Basu and Bultan studied a notion they also called synchronizability, but it differs from the notion studied in the present work; synchronizability and k-synchronizability define incomparable classes of communicating systems. The proofs of the decidability of synchronizability [3, 2] were shown to have flaws by Finkel and Lozes [9]. A question left open in their paper is whether synchronizability is decidable for mailbox communications, as originally claimed by Basu and Bultan. Akroun and Salaün defined also a property they called stability [1] and that shares many similarities with the synchronizability notion in [2].

Context-bounded model-checking is yet another approach for the automatic verification of concurrent systems. La Torre *et al.* studied systems of communicating machines extended with a calling stack, and showed that under some conditions on the interplay between stack actions and communications, context-bounded reachability was decidable [12]. A context-switch is found in an execution each time two consecutive actions are performed by a different participant. Thus, while k-synchronizability limits the number of consecutive sendings, bounded context-switch analysis limits the number of times two consecutive actions are performed by two different processes. It would be interesting to explore how both context-boundedness and communication-closed rounds could be composed.

References

[1] Lakhdar Akroun and Gwen Salaün. Automated verification of automata communicating via FIFO and bag buffers. *Formal Methods in System Design*, 52(3):260–276, 2018.

[2] Samik Basu and Tevfik Bultan. On deciding synchronizability for asynchronously communicating systems. *Theor. Comput. Sci.*, 656:60–75, 2016.

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

[4] Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 372–391. Springer, 2018.

[5] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.

[6] Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A reduction theorem for the verification of round-based distributed algorithms. In Olivier Bournez and Igor Potapov, editors, *Reachability Problems, 3rd International Workshop, RP 2009, Palaiseau, France, September 23-25, 2009. Proceedings*, volume 5797 of *Lecture Notes in Computer Science*, pages 93–106. Springer, 2009.

[7] Ching-Tsun Chou and Eli Gafni. Understanding and verifying distributed algorithms using stratified decomposition. In Danny Dolev, editor, *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 44–65. ACM, 1988.

[8] Tzilla Elrad and Nissim Francez. Decomposition of distributed programs into communication-closed layers. *Sci. Comput. Program.*, 2(3):155–173, 1982.

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

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

[11] Bernhard Kragl, Shaz Qadeer, and Thomas A. Henzinger. Synchronizing the asynchronous. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[12] Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. Context-bounded analysis of concurrent queue systems. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2008.

[13] Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Commun. ACM*, 18(12):717–721, 1975.

[14] Anca Muscholl. Analysis of communicating automata. In Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*, volume 6031 of *Lecture Notes in Computer Science*, pages 50–57. Springer, 2010.

[15] Klaus von Gleissenthall, Rami Gökhan Kici, Alexander Bakst, Deian Stefan, and Ranjit Jhala. Pretend synchrony: synchronous verification of asynchronous distributed programs. *PACMPL*, 3(POPL):59:1–59:30, 2019.