# Safe Implementability of Regular Message Sequence Chart Specifications

Nicolas BAUDRU & Rémi MORIN
Laboratoire d'Informatique Fondamentale de Marseille
Université de Provence, 39 rue F. Joliot-Curie, F-13453 Marseille cedex 13, France
baudru,morin@lif.univ-mrs.fr

## Abstract

*Message Sequence Charts (MSCs) are drawn by software designers in order to model the typical behaviors of some communication protocol at an early stage of its design. The important question of whether some given set of scenarios is realizable by some message passing system has already been investigated in different ways. We consider here deadlock-free implementations up to additional message contents. We present an algorithm to check safe implementability of MSC specifications in the framework of non-FIFO channels. Our criterion turns out to have the same complexity bounds as the restrictive approach of Alur et al. [1] where no additional message content is allowed.*

## 1. Introduction

Message Sequence Charts (MSCs) are a popular model often used for the documentation of telecommunication protocols. An MSC gives a graphical description of communications between processes along some particular scenario. It usually abstracts away from the values of variables and the actual contents of messages. Yet, this formalism can be used at a very early stage of design to detect errors in some specification. In this direction, several works have brought up methods for the model checking of high-level MSCs, see e.g. [2, 3, 6, 11, 15].

Recently, many articles have investigated the subclass of *regular* languages of MSCs [2, 3, 8, 9, 11, 13, 15]. These languages of MSCs are such that the set of all associated sequential executions can be described by a finite automaton; then, model checking becomes decidable and particular complexity results could be obtained [2, 3, 11, 15]. Popular, graphical, and powerful, MSCs are intuitive and easy to use. However they may lead to specifications that do not correspond to the operational behavior of any system of processes. The important question of whether a set of MSCs admits a realization has been investigated in different ways [1, 4, 7, 8]. In particular, [13, Th. 8] proved that any regular set of MSCs admits a *deterministic* implementation with bounded channel capacities up to some additional message contents called *time-stamps*. This result shows the power of adding contents to messages compared to the more restrictive approach proposed in [1].

Many classical distributed algorithms add stamps to message contents to solve inherent problems of asynchronous systems such as synchronization [10], back-up procedures [5], or logical clocks [12]. Adding contents to specified messages is also a natural counterpart to the fact that MSCs provide somewhat partial specifications that can abstract away from actual message contents [7, 8, 13]. In this view, the question arises to check whether there exists some *interpretation* of message contents so that the refined specification corresponds to the behavior of some finite state message passing system.

With the help of the notion of *coherence*, our first result characterizes the regular MSC specifications that can be implemented without deadlock (Th. 3.7). This criterion is proved by some synthesis algorithm that provides a safe implementation to any coherent regular set of MSCs. Our construction builds on known difficult results, namely Zielonka's theorem [17] and the bounded time-stamps of [14] in a new, modular, and abstract approach that avoids many technical details. Differently from [8] and similarly to [13], we consider here implementations that ensure a one-to-one correspondence between the observable execution sequences and the specified behaviors. Differently from [13], our construction allows for avoiding deadlocks even in the framework of non-FIFO channels.

Next we characterize which regular MSC specifications can be safely implemented *with respect to some local final states* (Th. 4.4). In this framework, we introduce two new interesting features called *definitive local stops* and absence of *spoiling message*. Similarly to Th. 3.7, our second criterion uses the notion of coherence that is easy to check from regular specifications. However, we observe that checking safe implementability is EXPSPACE-complete for the locally synchronized high-level MSCs of [3, 15] because of some classical state explosion problem (Th. 5.1). Due to the space limit, all proofs are omitted. Details are available in the forthcoming full paper.

## 2. Semantics of communicating systems

Let $\mathcal{I}$ be a finite set of processes (also called *instances*) and $\Lambda$ be a finite set of messages. For any instance $i \in \mathcal{I}$, the alphabet $\Sigma_i$ is the disjoint union of the set of *sending actions* $\Sigma_i^! = \{i!^m j \mid j \in \mathcal{I} \setminus \{i\}, m \in \Lambda\}$, the set of *receipt actions* $\Sigma_i^? = \{i?^m j \mid j \in \mathcal{I} \setminus \{i\}, m \in \Lambda\}$ and a finite set of internal actions $\Sigma_i^{int}$. We shall assume that the alphabets $\Sigma_i^{int}$ are disjoint and we let $\Sigma_{\mathcal{I}} = \bigcup_{i \in \mathcal{I}} \Sigma_i$. Given an action $a \in \Sigma_{\mathcal{I}}$, we denote by $\mathrm{Ins}(a)$ the unique instance $i$ such that $a \in \Sigma_i$, that is the particular instance on which each occurrence of action $a$ takes place.

In this paper, we consider systems of communicating automata where messages between two processes can overtake along non-FIFO channels — however overtaking of identical messages is meaningless in the present setting. Formally, the set of channels $\mathcal{K}$ consists of all triples $(i, j, m) \in \mathcal{I} \times \mathcal{I} \times \Lambda$ such that $i \neq j$. Furthermore a *channel state* is formalized by a map $\chi : \mathcal{K} \to \mathbb{N}$ that describes the queues of messages within the channels at some stage of an execution. The *empty channel state* $\chi_0$ is such that each channel maps to 0.

**Definition 2.1** *A* message-passing automaton (MPA) $\mathcal{S}$ *over* $\Sigma_{\mathcal{I}}$ *consists of a family of local components* $(\mathcal{A}_i)_{i \in \mathcal{I}}$ *and a subset of global final states* $F$ *such that each component* $\mathcal{A}_i$ *is a transition system* $(Q_i, \imath_i, \longrightarrow_i)$ *where* $Q_i$ *is a set of* $i$-local states, *with initial state* $\imath_i \in Q_i$, $\longrightarrow_i \subseteq (Q_i \times \Sigma_i \times Q_i)$ *is the* $i$-local transition relation *and* $F \subseteq \left(\prod_{i \in \mathcal{I}} Q_i\right) \times \mathbb{N}^{\mathcal{K}}$.

A *global state* is a pair $(s, \chi)$ where $s \in \prod_{i \in \mathcal{I}} Q_i$ is a tuple of local states and $\chi$ is a channel state. The *initial global state* is the pair $\imath = (s, \chi)$ such that $s = (\imath_i)_{i \in \mathcal{I}}$ and $\chi = \chi_0$ is the empty channel state. The *system of global states* associated to $\mathcal{S}$ is the transition system $\mathcal{A}_{\mathcal{S}} = (Q, \imath, \longrightarrow)$ where $Q = \prod_{i \in \mathcal{I}} Q_i \times \mathbb{N}^{\mathcal{K}}$ is the set of global states and the global transition relation $\longrightarrow \subseteq Q \times \Sigma_{\mathcal{I}} \times Q$ satisfies:

- for all $a \in \Sigma_i^{int}$, $(q_k)_{k \in \mathcal{I}}, \chi \xrightarrow{a} (q'_k)_{k \in \mathcal{I}}, \chi'$ if $\chi = \chi'$, $q_i \xrightarrow{a}_i q'_i$ and $q'_k = q_k$ for all $k \in \mathcal{I} \setminus \{i\}$;
- for all $(i, j, m) \in \mathcal{K}$, $(q_k)_{k \in \mathcal{I}}, \chi \xrightarrow{i!^m j} (q'_k)_{k \in \mathcal{I}}, \chi'$ if
  1. $q_i \xrightarrow{i!^m j}_i q'_i$ and $q'_k = q_k$ for all $k \in \mathcal{I} \setminus \{i\}$,
  2. $\chi'(i, j, m) = \chi(i, j, m) + 1$ and $\chi(x) = \chi'(x)$ for all $x \in \mathcal{K} \setminus \{(i, j, m)\}$;
- for all $(i, j, m) \in \mathcal{K}$, $(q_k)_{k \in \mathcal{I}}, \chi \xrightarrow{j?^m i} (q'_k)_{k \in \mathcal{I}}, \chi'$ if
  1. $q_j \xrightarrow{j?^m i}_j q'_j$ and $q'_k = q_k$ for all $k \in \mathcal{I} \setminus \{j\}$,
  2. $\chi(i, j, m) = 1 + \chi'(i, j, m)$ and $\chi(x) = \chi'(x)$ for all $x \in \mathcal{K} \setminus \{(i, j, m)\}$.

As usual with transition systems, for any $u = a_1 ... a_n \in \Sigma_{\mathcal{I}}^\star$, we write $q \xrightarrow{u} q'$ if there are some global states $q_0, ..., q_n \in Q$ such that $q_0 = q$, $q_n = q'$ and for all $r \in [1, n]$, $q_{r-1} \xrightarrow{a_r} q_r$. An *execution sequence* of $\mathcal{S}$ is a word $u \in \Sigma_{\mathcal{I}}^\star$ such that $\imath \xrightarrow{u} q$ for some global state $q$.

An execution sequence $u$ is a *final execution sequence* if $\mathcal{S}$ accepts $u$, i.e. $\imath \xrightarrow{u} q$ for some final global state $q \in F$. In this paper, following [8, 9, 13], we allow the specification of global final states $F$. The latter are usually chosen with empty channel states. Another approach is to provide each instance with a subset of local final states $F_i \subseteq Q_i$. Then a global state $q = ((q_i)_{i \in \mathcal{I}}, \chi)$ is considered final if each local state $q_i$ is a final local state and the channel state $\chi$ is empty [1, 2, 6, 11]. We will adopt this restriction in Section 4.

Following [1, 2], we call *deadlock* any reachable global state from which no final global state is reachable. Another approach to take the notion of deadlock into account was followed in [7] which assumes that all states are final.

**Message sequence charts are labelled partial orders.** As usual in models of concurrency, the behavior of an MPA can be described by labelled partial orders [10, 17]. A *pomset* over an alphabet $\Sigma$ is a triple $t = (E, \preccurlyeq, \xi)$ where $(E, \preccurlyeq)$ is a finite partial order and $\xi$ is a mapping from $E$ to $\Sigma$ such that $\xi(x) = \xi(y)$ implies $x \preccurlyeq y$ or $y \preccurlyeq x$. Let $t = (E, \preccurlyeq, \xi)$ be a pomset and $x, y \in E$. Then $y$ *covers* $x$ (denoted $x \prec\!\!\cdot\, y$) if $x \prec y$ and $x \prec z \preccurlyeq y$ implies $y = z$. A pomset can be seen as a model for an execution of a concurrent system: In this view, the elements $e$ of $E$ are *events* and their label $\xi(e)$ describes the basic action of the system that is performed by the event. Furthermore, the order describes the causal dependence between the events. An *order extension* of a pomset $t = (E, \preccurlyeq, \xi)$ is a pomset $t' = (E, \preccurlyeq', \xi)$ such that $\preccurlyeq \subseteq \preccurlyeq'$. A *linear extension* of $t$ is an order extension that is linearly ordered. Linear extensions of a pomset $t = (E, \preccurlyeq, \xi)$ can naturally be regarded as words over $\Sigma$. By $\mathrm{LE}(t) \subseteq \Sigma^\star$, we denote the set of linear extensions of a pomset $t$ over $\Sigma$. An *ideal* of a pomset $t = (E, \preccurlyeq, \xi)$ is a subset $H \subseteq E$ which is downward-closed: $x \in H \wedge y \preccurlyeq x \Rightarrow y \in H$. The restriction $t' = (H, \preccurlyeq \cap (H \times H), \xi \cap (H \times \Sigma))$ is called a *prefix* of $t$ and we write $t' \leqslant t$. For all $z \in E$, we denote by $\downarrow z$ the ideal of events below $z$, i.e. $\downarrow z = \{y \in E \mid y \preccurlyeq z\}$. If $H$ is a subset of $E$, we denote by $|H|_a$ the number of events $x \in H$ such that $\xi(x) = a$. For any set of pomsets $\mathcal{L}$, we write $\mathrm{Pref}(\mathcal{L})$ for the set of all prefixes of pomsets in $\mathcal{L}$.

Consider now a pomset $(E, \preccurlyeq, \xi)$ over the fixed alphabet $\Sigma_{\mathcal{I}}$. We denote by $\mathrm{Ins}(e)$ the instance on which the event $e \in E$ occurs : $\mathrm{Ins}(e) = \mathrm{Ins}(\xi(e))$. Message Sequence Charts (MSCs for short) are defined by several recommendations that indicate how one should represent them graphically. Examples of such drawings are given in Fig. 1 and Fig. 3. These drawings must be read top-down. Formally, MSCs can be seen simply as particular pomsets over $\Sigma_{\mathcal{I}}$.

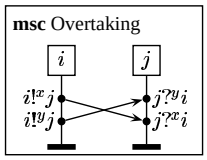


FIG. 1. Overtaking

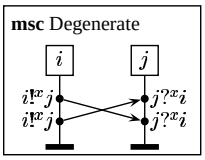


FIG. 2. Degenerate MSC

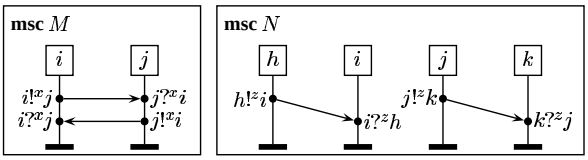


FIG. 3. Basic MSCs $M$ and $N$

**Definition 2.2** *A* message sequence chart (MSC) *is a pomset* $M = (E, \preccurlyeq, \xi)$ *over* $\Sigma_{\mathcal{I}}$ *such that for all* $e, f \in E$,

$\mathsf{M}_1$: *if* $\mathrm{Ins}(e) = \mathrm{Ins}(f)$ *then* $e \preccurlyeq f$ *or* $f \preccurlyeq e$;

$\mathsf{M}_2$: $|E|_{j?^m i} \leqslant |E|_{i!^m j}$ *for all channels* $(i, j, m) \in \mathcal{K}$;

$\mathsf{M}_3$: *if* $\xi(e) = i!^m j$, $\xi(f) = j?^m i$, *and* $|{\downarrow}e|_{i!^m j} = |{\downarrow}f|_{j?^m i}$ *then* $e \preccurlyeq f$;

$\mathsf{M}_4$: *if* $e \prec\!\!\!\cdot\, f$ *and* $\mathrm{Ins}(e) \neq \mathrm{Ins}(f)$ *then* $\xi(e) = i!^m j$, $\xi(f) = j?^m i$, *and* $|{\downarrow}e|_{i!^m j} = |{\downarrow}f|_{j?^m i}$.

*An MSC is* basic *if* $|E|_{i!^m j} = |E|_{j?^m i}$ *for all* $(i, j, m) \in \mathcal{K}$.

By $\mathsf{M}_1$, events occurring on the same instance are linearly ordered: Non-deterministic choice cannot be described within an MSC. Condition $\mathsf{M}_2$ makes sure that each receipt corresponds to a sending event. There is no duplication nor loss of messages in the channels and $\mathsf{M}_2$ formalizes partly the reliability of the channels. Yet non-FIFO behavior is allowed because there is a channel between every pair of processes and for every message: In other words, we allow *overtaking* (Fig. 1). However we forbid any reversal of the order in which two *identical* messages $m$ sent from $i$ to $j$ are received by $j$ (Fig. 2). This restriction allows us to recover the correspondence between sending and receipt events from the partial order: In particular, $\mathsf{M}_3$ formalizes that the receipt of any message will occur after the corresponding sending event. Finally, by $\mathsf{M}_4$, causality in $M$ consists only in the linear dependence over each instance and the ordering of pairs of corresponding sending and receipt events. In a basic MSC, all messages sent are received.

The *communication graph* of an MSC $M = (E, \preccurlyeq, \xi)$ is the directed graph $(\mathcal{I}_M, \mapsto)$ where $\mathcal{I}_M$ is the set of *active instances* of $M$: $\mathcal{I}_M = \{i \in \mathcal{I} \mid \exists e \in E, \mathrm{Ins}(e) = i\}$, and such that $(i, j) \in \mapsto$ if there is an event $e \in E$ such that $\xi(e) = i!^m j$. Thus there is an edge from $i$ to $j$ if $M$ shows a communication from $i$ to $j$. An MSC is called *connected* (resp. strongly connected) if its communication graph is connected (resp. strongly connected). As we will see below, communication graphs are useful to classify high-level MSCs.

**HMSCs as rational expressions.** We denote by $\mathrm{bMSC}$ the set of basic MSCs. The *asynchronous concatenation* of two basic MSCs $M_1 = (E_1, \preccurlyeq_1, \xi_1)$ and $M_2 = (E_2, \preccurlyeq_2, \xi_2)$ is $M_1 \cdot M_2 = (E, \preccurlyeq, \xi)$ where $E = E_1 \uplus E_2$, $\xi = \xi_1 \cup \xi_2$ and the partial order $\preccurlyeq$ is the transitive closure of $\preccurlyeq_1 \cup \preccurlyeq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \mathrm{Ins}(e_1) = \mathrm{Ins}(e_2)\}$. It is easy to check that the asynchronous concatenation of two basic MSCs is a basic MSC. This concatenation enables us to compose specifications in order to describe sets of basic MSCs. Now high-level message sequence charts may be defined as rational expressions, following the usual algebraic approach that we recall next.

For any subsets $\mathcal{L}$ and $\mathcal{L}'$ of $\mathrm{bMSC}$, the *product* of $\mathcal{L}$ by $\mathcal{L}'$ is $\mathcal{L} \cdot \mathcal{L}' = \{x \cdot x' \mid x \in \mathcal{L} \wedge x' \in \mathcal{L}'\}$. We let $\mathcal{L}^0 = \{1\}$ and for any $n \in \mathbb{N}$, $\mathcal{L}^{n+1} = \mathcal{L}^n \cdot \mathcal{L}$; then the *iteration* of $\mathcal{L}$ is $\mathcal{L}^\star = \bigcup_{n \in \mathbb{N}} \mathcal{L}^n$.

**Definition 2.3** *A* high-level message sequence chart (HMSC) *is a rational expression of basic MSCs, that is, an expression built from finite sets of basic MSCs by use of union* $(+)$*, product* $(\cdot)$ *and iteration* $(\star)$.

We follow here the approach adopted, e.g., in [2, 4, 11, 13, 15] where HMSCs are however often flattened into *message sequence graphs*. The set of MSCs corresponding to some HMSC $\mathcal{H}$ is denoted by $\mathcal{L}_{\mathcal{H}}$.

Rational expressions of MSCs can describe unexpected behaviors: For instance, HMSCs can show process divergence, i.e. unbounded numbers of messages within some channels [13]. The *channel-width* of an MSC $M$ is

$$\max_{(i,j,m) \in \mathcal{K}} \{|v|_{i!^m j} - |v|_{j?^m i} \mid v \leqslant u \wedge u \in \mathrm{LE}(M)\}.$$

Thus, the channel-width of $M$ is the maximal number of messages that may be sent in a channel but not yet received along some linear execution of $M$. A language $\mathcal{L} \subseteq \mathrm{bMSC}$ is *channel-bounded* by an integer $B$ if each basic MSC of $\mathcal{L}$ has a channel-width at most $B$. Let us consider for instance the basic MSCs $M$ and $N$ of Fig. 3. The HMSC $\{M\}^\star$ is channel-bounded by 1 whereas $\{N\}^\star$ is not channel-bounded.

Regular languages of basic MSCs were defined in [8] as subsets $\mathcal{L}$ of $\mathrm{bMSC}$ such that the corresponding set of linear extensions $\mathrm{LE}(\mathcal{L})$ is a regular set of words in $\Sigma_{\mathcal{I}}^\star$. It was also observed that each regular language is channel-bounded. However regularity of HMSCs is undecidable [9, Th. 4.6]. A HMSC is called *locally synchronized* [3, 15] (resp. *globally cooperative* [6]) if iteration $\star$ occurs only over sets of *strongly connected* (resp. connected) MSCs. As observed in [15, 3], locally synchronized HMSCs describe regular sets of MSCs.

**Implementation of MSC languages.** Consider now an MPA $\mathcal{S}$ with components $(\mathcal{A}_i)_{i \in \mathcal{I}}$ and global final states $F$. Let $u \in \Sigma_{\mathcal{I}}^\star$ be an execution sequence with $\imath \xrightarrow{u} q$

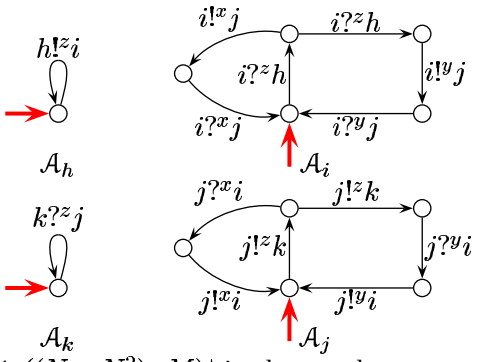


FIG. 4. $((N + N^2) \cdot M)^\star$ implemented up to $x, y \mapsto x$

for some global state $q$. Then $u$ is a linear extension of a unique MSC. The latter is a *basic* MSC if, and only if, $q$ has an empty channel-state. The *language of basic MSCs* $\mathcal{L}(\mathcal{S})$ consists of the basic MSCs $M$ such that at least one linear extension of $M$ is a final execution sequence of $\mathcal{S}$. Noteworthy, it can be easily shown that a basic MSC belongs to $\mathcal{L}(\mathcal{S})$ iff *all* its linear extensions are final execution sequences of $\mathcal{S}$. We say that an MPA $\mathcal{S}$ *with finitely many local states* is a *realization* of some language of basic MSCs $\mathcal{L}$ if $\mathcal{L}(\mathcal{S}) = \mathcal{L}$.

Let us look at two examples. First, consider the basic MSCs $M$ and $N$ of Fig. 3. We can show that $\mathcal{H} = ((N + N^2) \cdot M)^\star$ admits no realization at all. However, we can build an MPA that accepts an *interpretation* of $\mathcal{H}$. To see this, let $M'$ be the basic MSC obtained from $M$ by replacing message $x$ by message $y$. Then the MPA $\mathcal{S}_1$ with four components depicted on Fig. 4 is a realization of the language $\mathcal{H}' = (N \cdot M + N^2 \cdot M')^\star$ when the set of global final state is restricted to the initial state marked by bold arrows on the picture. Thus $\mathcal{S}_1$ is a realization of $\mathcal{H}$ *up to the identification of messages $x$ and $y$.* This approach was investigated in [8] and adopted also in [13, 7, 6].

**Definition 2.4** *An* interpretation *of a language $\mathcal{L}$ of MSCs over $\Lambda$ is a language of MSCs $\mathcal{L}'$ over $\Lambda'$ and a mapping $\gamma$ from $\Lambda'$ onto $\Lambda$ such that $\gamma$ induces a one-to-one correspondence from the prefixes of $\mathcal{L}'$ onto the prefixes of $\mathcal{L}$.*

*We say that $\mathcal{L}$ is* implementable *if there exists some interpretation $\mathcal{L}'$ of $\mathcal{L}$ such that $\mathcal{L}'$ admits a realization.*

We stress here that each prefix of an MSC of $\mathcal{L}$ corresponds to a unique prefix of an MSC of $\mathcal{L}'$. This requirement ensures that interpretations will preserve and reflect coherence (Def. 3.1 below). This condition was adopted in [13], improving the results established in [8]. Thus our notion of implementability includes determinism in the sense of [13], whose main result can be stated as follows.

**Theorem 2.5** *[13, Th. 3.2] All regular sets of basic MSCs are implementable.*

This result and the above example show the power of refining message contents in order to help the realization of sets of scenarios. As opposed to the restrictive approach studied in [1, 2, 11] which sticks to the specified set of message contents, interpretations allow for the implementation of any finite set of basic MSCs. Theorem 2.5 was actually established in [13] under some FIFO assumption. However the construction presented in the next section will show that this result holds in presence of overtaking, too.

## 3. Deadlock-free non-FIFO implementations

In this section, we characterize the regular MSC languages that are implementable without deadlock. This is based on the notion of coherence that we define next.

**Definition 3.1** *Let $\mathcal{L}$ be a set of basic MSCs and $L$ be the set of prefixes of its linear extensions* $\mathrm{LE}(\mathcal{L})$. *Then $\mathcal{L}$ is said* coherent *if for all $u \in \Sigma_I^\star$ and all $a, b \in \Sigma_I$:*

$$u.a \in L \wedge u.b \in L \wedge \mathrm{Ins}(a) \neq \mathrm{Ins}(b) \Rightarrow u.ab \in L$$

Intuitively, whenever two actions can occur on two distinct processes then both can occur independently along an accepting execution. Clearly, if an MPA $\mathcal{S}$ is deadlock-free then its language $\mathcal{L}(\mathcal{S})$ is coherent. Furthermore if a language of basic MSCs $\mathcal{L}$ is implementable by an MPA without deadlock then $\mathcal{L}$ is coherent. We show here the converse property (Th. 3.7): We build a safe implementation of any coherent regular MSC language. For this, similarly to [8], we use some results from Mazurkiewicz trace theory [17, 16] together with a communication protocol detailed in [14]. Differently from [13], this modular approach allows us to abstract from many technical difficulties. Differently from [8], our construction avoids nondeterminism and deadlock. We stress also that we cannot simply adapt the construction done in [13] because we deal here with non-FIFO channels.

**Basic notions in Mazurkiewicz trace theory.** The concurrency of a distributed system can be represented by an *independence relation* over the alphabet of actions $\Sigma$, that is a binary, symmetric and irreflexive relation $\| \subseteq \Sigma \times \Sigma$. The associated *trace equivalence* is the least congruence $\sim$ over $\Sigma^\star$ such that $\forall a, b \in \Sigma, a \| b \Rightarrow ab \sim ba$. A *trace* $[u]$ is the equivalence class of a word $u \in \Sigma^\star$. We denote by $\mathbb{M}(\Sigma, \|)$ the set of all traces w.r.t. $(\Sigma, \|)$. A *trace language* is a subset $\mathcal{L} \subseteq \mathbb{M}(\Sigma, \|)$. It is called regular if the set of associated words $\{u \in \Sigma^\star \mid [u] \in \mathcal{L}\}$ is recognizable in the free monoid $\Sigma^\star$, i.e. described by a finite automaton.

Let $u \in \Sigma^\star$; then the trace $[u]$ is precisely the set of linear extensions $\mathrm{LE}(t)$ of a unique pomset $t = (E, \preccurlyeq, \xi)$, that is, $[u] = \mathrm{LE}(t)$. Moreover $t$ satisfies the following additional properties:

$\mathsf{MP}_1$: $\forall e_1, e_2 \in E, \xi(e_1) \not\| \xi(e_2) \Rightarrow (e_1 \preccurlyeq e_2 \vee e_2 \preccurlyeq e_1)$;
$\mathsf{MP}_2$: $\forall e_1, e_2 \in E, e_1 \mathrel{—\!\prec} e_2 \Rightarrow \xi(e_1) \not\| \xi(e_2)$.

Conversely the linear extensions of a pomset satisfying these two axioms form a trace of $\mathbb{M}(\Sigma, \|)$. That is why

one usually identifies $\mathbb{M}(\Sigma,||)$ with the class of pomsets satisfying $\mathsf{MP}_1$ and $\mathsf{MP}_2$. Thus traces can be regarded as pomsets and we put $\mathrm{LE}(\mathcal{L}) = \{u \in \Sigma^\star \mid [u] \in \mathcal{L}\}$ for any trace language $\mathcal{L} \subseteq \mathbb{M}(\Sigma,||)$.

Our approach starts with a strong relationship between MSCs and traces that appears when we consider MSCs of channel-width at most 1.

**Lemma 3.2** *Let $||_1$ be the independence relation over $\Sigma_{\mathcal{I}}$ such that $a \not||_1 b$ if $\mathrm{Ins}(a) = \mathrm{Ins}(b)$ or $\{a,b\} = \{i!^m j, j?^m i\}$ for some channel $(i,j,m) \in \mathcal{K}$. An MSC $M$ is a Mazurkiewicz trace of $\mathbb{M}(\Sigma_{\mathcal{I}},||_1)$ if $M$ has channel-width at most 1.*

Consider for instance the MSC $N$ of Fig. 3. Then $N^2$ has channel-width 2 and it is not a trace of $\mathbb{M}(\Sigma_{\mathcal{I}},||_1)$.

This connection is quite useful because we can interpret any channel-bounded language of MSCs as a set of MSCs of channel-width at most 1 using a second basic observation.

**Lemma 3.3** *Let $B$ be a positive integer and $\Sigma_{\mathcal{I}}^B$ be the alphabet induced by the extended set of messages $\Lambda \times [0, B-1]$. Let $M = (E, \preccurlyeq, \xi)$ be an MSC with channel-width at most $B$. Then the pomset $M^\dagger = (E, \preccurlyeq, \xi^\dagger)$ over $\Sigma_{\mathcal{I}}^B$ where*
*$\xi^\dagger(e) = \xi(e)$ if $\xi(e)$ is an internal action,*
*$\xi^\dagger(e) = i!^{m,n} j$ if $\xi(e) = i!^m j \wedge n = |{\downarrow}e|_{i!^m j} \mod B$,*
*$\xi^\dagger(e) = j?^{m,n} i$ if $\xi(e) = j?^m i \wedge n = |{\downarrow}e|_{j?^m i} \mod B$,*
*is an MSC with channel-width at most 1.*

Thus, any regular language of basic MSCs can be interpreted as a regular set of traces (Lemma 3.2). This interpretation preserves coherence (Def. 3.1) when we adopt the following definition.

**Definition 3.4** *Let $\mathcal{L} \subseteq \mathbb{M}(\Sigma,||)$ be a set of traces and $L$ be the set of prefixes of $\mathrm{LE}(\mathcal{L})$. Then $\mathcal{L}$ is said* coherent *if for all $u \in \Sigma^\star$ and all $a, b \in \Sigma$:*

$$u.a \in L \wedge u.b \in L \wedge a||b \Rightarrow u.ab \in L$$

**Implementation of Mazurkiewicz trace languages.** Let $(\Sigma,||)$ be an independence alphabet and $K$ a set of processes. We fix a covering by cliques $\rho = (\Delta_k)_{k\in K}$ of $(\Sigma, \not||)$: For all $a, b \in \Sigma$, $a \not|| b$ if and only if $\{a,b\} \subseteq \Delta_k$ for some $k \in K$. For each action $a \in \Sigma$, we put $\mathrm{Loc}(a) = \{k \in K \mid a \in \Delta_k\}$.

An *asynchronous automaton* $\mathcal{A}$ consists of a family of finite sets of local states $(Q_k)_{k\in K}$, an initial local state $\imath_k$ for each process $k \in K$, and for each action $a \in \Sigma$ a transition function $\delta_a : \prod_{k\in \mathrm{Loc}(a)} Q_k \to \prod_{k\in \mathrm{Loc}(a)} Q_k$ *partially defined*. The set of global states is $Q = \prod_{k\in K} Q_k$, the global initial state is $\imath = (\imath_k)_{k\in K}$, and the global transition relation $\longrightarrow \subseteq Q \times \Sigma \times Q$ is such that a $a$-transition leads from $(q_k)_{k\in K}$ to $(q'_k)_{k\in K}$ if $\delta_a((q_k)_{k\in \mathrm{Loc}(a)}) = (q'_k)_{k\in \mathrm{Loc}(a)}$ and $q_k = q'_k$ for all $k \notin \mathrm{Loc}(a)$.

A basic observation is that for all words $u, v \in \Sigma^\star$, if $q \xrightarrow{u} q'$ and $v \sim u$ then $q \xrightarrow{v} q'$. For any subset of final global states $F \subseteq Q$, the trace language recognized by $(\mathcal{A}, F)$ consists of all traces $[u]$ such that $\imath \xrightarrow{u} q$ for some final global state $q \in F$. Now, a global state $q \in Q$ of an asynchronous automaton is called a *deadlock* if $q$ is reachable from the initial state but no final state is reachable from $q$.

Since there are finitely many global states, the trace language recognized by some asynchronous automaton is regular. A celebrated result from trace theory due to Zielonka shows the converse property [17]. A refinement of Zielonka's theorem was used recently in [16]. It allows for building asynchronous automata without deadlock when implementing coherent trace languages.

**Theorem 3.5** *Let $\mathcal{L} \subseteq \mathbb{M}(\Sigma,||)$ be a regular and coherent set of traces. There exists some asynchronous automaton $(\mathcal{A}, F)$* without deadlock *that recognizes $\mathcal{L}$.*

**Communication protocol.** We introduce some notations useful before applying [14]. Let $M = (E, \preccurlyeq, \xi)$ be an MSC. We denote by $\max_i(M)$ the latest event run by the process $i$, i.e. $\max_i(M) = e$ if $\mathrm{Ins}(e) = i \wedge \mathrm{Ins}(e') = i$ implies $e' \preccurlyeq e$ for all $e' \in E$. Now, $P_{i,j}(M)$ represents the longest execution run by the process $j$ and seen by the process $i$, i.e. $P_{i,j}(M) = (E_j \cap {\downarrow}\max_i(M), \preccurlyeq, \xi)$ where $E_j$ is the set of events located on process $j$. Since this restriction of $M$ is a total order over $E_j$, it can be identified with a word of $\Sigma_j^\star$. Let $\mathcal{S} = (\mathcal{A}_i)_{i\in\mathcal{I}}$ be an MPA and $s = \imath \xrightarrow{a_1} q_1 ... q_{n-1} \xrightarrow{a_n} q_n$ be a global computation of $\mathcal{S}$ such that $a_1 ... a_n \in \mathrm{LE}(M)$ for the MSC $M$. For $i, j \in \mathcal{I}$, we denote by $F_{i,j}(s)$ the local state reached by $\mathcal{A}_j$ along the computation $s$ after executing $P_{i,j}(M)$. The channel-width of $s$ is the channel-width of the associated MSC $M$.

Let $\Lambda^\dagger$ be a new set of messages and $\Sigma_{\mathcal{I}}^\dagger$ be the alphabet induced by $\Lambda \times \Lambda^\dagger$ and the same set of internal actions. We denote by $\pi_1$ the projection from $\Lambda \times \Lambda^\dagger$ to $\Lambda$. By abuse of notations, we also denote by $\pi_1$ the mapping from $\Sigma_{\mathcal{I}}^\dagger$ to $\Sigma_{\mathcal{I}}$ where all messages $(m, c)$ loose their additional part $c \in \Lambda^\dagger$. This notation extends in a natural way to words over $\Sigma_{\mathcal{I}}^\dagger$ and MSCs over $\Lambda \times \Lambda^\dagger$. In the sequel of this section, we shall consider an MPA $\mathcal{S}$ with global final states $F$ and components $\mathcal{A}_i = (Q_i, \imath_i, \longrightarrow_i)$ over the set of messages $\Lambda$. A *stamping MPA* of $\mathcal{S}$ is an MPA $\mathcal{S}^\dagger$ over the set of messages $\Lambda \times \Lambda^\dagger$ whose components $\mathcal{A}_i^\dagger = (Q_i \times R_i, (\imath_i, r_i^\circ), \longrightarrow_{\dagger,i})$ satisfy the following requirements:

- $\mathsf{S}_1$: if $(q,r) \xrightarrow{a}_{\dagger,i} (q',r')$ and $(q,r) \xrightarrow{a}_{\dagger,i} (q'',r'')$ then $q' = q''$ and $r' = r''$;
- $\mathsf{S}_2$: if $(q,r) \xrightarrow{a}_{\dagger,i} (q',r')$ then $q \xrightarrow{\pi_1(a)}_i q'$;
- $\mathsf{S}_3$: if $q \xrightarrow{a}_i q'$, $a \in \Sigma_i^{int}$, and $r \in R_i$ then $(q,r) \xrightarrow{a}_{\dagger,i} (q',r')$ for some $r' \in R_i$;
- $\mathsf{S}_4$: if $q \xrightarrow{a}_i q'$, $a = i!^m j$, and $r \in R_i$ then there exists a unique $c \in \Lambda^\dagger$ such that $(q,r) \xrightarrow{i!^{m,c} j}_{\dagger,i} (q',r')$ for some $r' \in R_i$;
- $\mathsf{S}_5$: if $q \xrightarrow{a}_i q'$, $a = i?^m j$, and $r \in R_i$ then for all $c \in \Lambda^\dagger$, $(q,r) \xrightarrow{i?^{m,c} j}_{\dagger,i} (q',r')$ for some $r' \in R_i$.

A global state $((q_i,r_i)_{i\in\mathcal{I}},\chi_0)$ is final if $((q_i)_{i\in\mathcal{I}},\chi_0)$ is a final global state of $\mathcal{S}$.

The requirements $\mathsf{S}_1$ to $\mathsf{S}_5$ ensure that for any MSC $M\in\mathcal{L}(\mathcal{S})$ there is some execution $M^\dagger$ in the stamping MPA $\mathcal{S}^\dagger$ such that $\pi_1(M^\dagger)=M$. Conversely, consider an execution $M^\dagger$ of $\mathcal{L}(\mathcal{S}^\dagger)$ such that two messages $(m,c)$ and $(m,c')$ are never simultaneously in transit from $i$ to $j$. Then the projected pomset $\pi_1(M^\dagger)$ is an MSC; moreover it is an execution of $\mathcal{S}$. The problem raised by two messages $(m,c)$ and $(m,c')$ simultaneously in transit from $i$ to $j$ is that these messages can overtake in $\mathcal{S}^\dagger$ leading to some degeneracy in $\pi_1(M^\dagger)$. This is the reason why we must consider specifications that are channel-bounded by 1 up to some preliminary interpretation.

We use here the crucial time-stamping protocol of [14] which can be applied in our framework as follows.

**Theorem 3.6** *Let $\mathcal{S}=(\mathcal{A}_i)_{i\in\mathcal{I}}$ be an MPA over $\Lambda$, and $B$ be a positive integer. There are a finite alphabet $\Lambda^\dagger$, a stamping MPA $\mathcal{S}^\dagger$ of $\mathcal{S}$ over the set of messages $\Lambda\times\Lambda^\dagger$ with components $\mathcal{A}_i^\dagger=(Q_i\times R_i,(\iota_i,r_i^\circ),\longrightarrow_{\dagger,i})$, and a family of maps $(\Pi_{i,j}:R_i\to Q_j)_{i,j\in\mathcal{I}}$ such that for any computation $s$ that leads the stamping MPA $\mathcal{S}^\dagger$ from its initial state to the global state $(q_i,r_i)_{i\in\mathcal{I}}$, if $\pi_1(s)$ has a channel-width at most $B$ then $\Pi_{i,j}(r_i)=F^{\mathcal{S}}_{i,j}(\pi_1(s))$ for all $i,j\in\mathcal{I}$.*

Intuitively, the stamps added by $\mathcal{S}^\dagger$ allow to compute and update a correct snapshot of $F_{i,j}(s)$ executed in $\mathcal{S}$.

**Synthesis algorithm.** Consider now a coherent and regular language of basic MSCs $\mathcal{L}\subseteq\mathrm{bMSC}$ with set of messages $\Lambda$. We show here how to build a safe implementation of $\mathcal{L}$. By Lemma 3.3, we may assume that $\mathcal{L}$ is channel-bounded by 1. By Lemma 3.2, $\mathcal{L}$ appears as a regular and coherent set of traces over the independence alphabet $(\Sigma_{\mathcal{I}},||_1)$. We apply Th. 3.5 and get an asynchronous automaton $Z$ that recognizes the trace language $\mathcal{L}$ with the clique covering $\rho=(\Delta_i)_{i\in\mathcal{I}}\cup(\Delta_{i,j,m})_{(i,j,m)\in\mathcal{K}}$ where $\Delta_i=\Sigma_i$ and $\Delta_{i,j,m}=\{i!^m j,j?^m i\}$. Thus, the set of processes is $K=\mathcal{I}\uplus\mathcal{K}$. We denote by $X_i$ the local states of process $i$ and by $X_{i,j,m}$ those of $(i,j,m)$. We put $X_{\mathcal{I}}=\bigcup_{i\in\mathcal{I}}X_i$.

From $Z$ and its transition functions $\delta_a$, we build an MPA $\mathcal{S}=(\mathcal{A}_i)_{i\in\mathcal{I}}$ over the set of messages $\Lambda\times X_{\mathcal{I}}$ as follows. Let $Y_i$ be the set of all channels $(k,l,m)\in\mathcal{K}$ such that $i=k$ or $i=l$. For any $i\in\mathcal{I}$, the local component $\mathcal{A}_i$ is such that a local state is of $\mathcal{A}_i$ is any tuple $(x,(x_{j,m})_{j,m\in\mathcal{I}\setminus\{i\}\times\Lambda},\zeta)$ where $x\in X_i$, $x_{j,m}\in X_{j,i,m}$, and $\zeta$ is a map from $Y_i$ to $\{0,1\}$. Intuitively, $\zeta$ describes the state of the channels related to $i$ and $x_{j,m}$ simulates the process $(j,i,m)$. The initial state is the tuple $(x,(x_{j,m})_{j,m\in\mathcal{I}\setminus\{i\}\times\Lambda},\zeta)$ where $x$ is the initial state of process $i$, $x_{j,m}$ is the initial state of process $(j,i,m)$, and $\zeta=0$. For all local states $q=(x,(x_{k,n})_{k,n\in\mathcal{I}\setminus\{i\}\times\Lambda},\zeta)$ and $q'=(x',(x'_{k,n})_{k,n\in\mathcal{I}\setminus\{i\}\times\Lambda},\zeta')$, for all messages $(m,w)\in\Lambda\times X_{\mathcal{I}}$, and for all instances $j\in\mathcal{I}$, we put $q\xrightarrow{i!^{m,w}j}_i q'$ if $x=w$, $\zeta'(i,j,m)=1+\zeta(i,j,m) \mod 2$, there is a transition in $Z$ such that $\delta_{i!^m j}(x,z)=(x',z')$, and the other components of $q'$ are identical to those of $q$. Moreover we put $q\xrightarrow{i?^{m,w}j}_i q'$ if $\zeta'(j,i,m)=1+\zeta(j,i,m) \mod 2$, there are two transitions in $Z$ such that $\delta_{j!^m i}(w,x_{j,m})=(w',z)$ and $\delta_{i?^m j}(x,z)=(x',x'_{j,m})$, and the other components of $q'$ are identical to those of $q$. Finally $q\xrightarrow{a}q'$ for some internal action $a\in\Sigma_i^{int}$ if $\delta_a(x)=x'$ and the other components of $q'$ are identical to those of $q$.

Note here that if $\mathcal{A}_i$ has a local transition $q\xrightarrow{i!^{m,w}j}_i q'$ then $w\in X_i$. Intuitively, when a message is sent from $i$ to $j$, the local state of $\mathcal{A}_i$ at this point is added to the message content. This additional information will allow the receiver $\mathcal{A}_j$ to simulate the behavior of the channel $(i,j,m)$ in two steps when receiving this message. Also, when sending a message $(m,w)$ to $\mathcal{A}_j$, instance $\mathcal{A}_i$ guesses the current state $z$ of process $(i,j,m)$. Similarly to the construction of [8], this guess produces non-determinism in $\mathcal{A}_i$ — and possible deadlocks in $\mathcal{S}$. However, this non-determinism will disappear later in our construction by considering a new component $\mathcal{A}_i^\ddagger$ instead of $\mathcal{A}_i$. To complete the definition of $\mathcal{S}=(\mathcal{A}_i)_{i\in\mathcal{I}}$, a global state $(x_i,(x_{j,i,m})_{j,m\in\mathcal{I}\setminus\{i\}\times\Lambda},\zeta_i)_{i\in\mathcal{I}}$ with empty channel-state is final if $((x_i)_{i\in\mathcal{I}},(x_{i,j,m})_{i,j,m\in\mathcal{K}})$ is a final state of $Z$.

We apply now Theorem 3.6 on $\mathcal{S}$ with $B=1$ and get a stamping MPA $\mathcal{S}^\dagger$ such that for any computation $s$ of $\mathcal{S}^\dagger$, each local component $\mathcal{A}_i^\dagger$ of $\mathcal{S}^\dagger$ knows the local state $F^{\mathcal{S}}_{i,j}(\pi_1(s))$ of each component $\mathcal{A}_j$, provided that $\pi_1(s)$ has channel-width at most 1. In order to avoid non-determinism and deadlocks, we will use $F^{\mathcal{S}}_{i,j}(\pi_1(s))$ in order to forbid some local transitions of $\mathcal{A}_i^\dagger$. We get in that way new components $\mathcal{A}_i^\ddagger$ and a corresponding MPA $\mathcal{S}^\ddagger$. These restrictions on $\mathcal{A}_i^\dagger$ will also ensure that no more than one message $m\in\Lambda$ stays in the channels from $i$ to $j$, i.e. $\sum_{x,c\in X_{\mathcal{I}}\times\Lambda^\dagger}\chi(i,j,m,x,c)\leqslant 1$ for all reachable states $(q,\chi)$ of $\mathcal{S}^\ddagger$. Consequently, $s$ and $\pi_1(s)$ have channel-width at most 1 for any computation $s$ of $\mathcal{S}^\ddagger$. This is done by comparing the counters $\zeta(i,j,m)$ on $i$ and $j$ before sending a new message $m$ to $j$. Now, the rules for forbidding local transitions of $\mathcal{A}_i^\dagger$ are the following. Given two local states $q=(x,(x_{j,m})_{j,m\in\mathcal{I}\setminus\{i\}\times\Lambda},\zeta)$ and $q'=(x',(x'_{j,m})_{j,m\in\mathcal{I}\setminus\{i\}\times\Lambda},\zeta')$ of $\mathcal{A}_i$,

- every local transition $(q,r)\xrightarrow{a}_{\dagger,i}(q',r')$ in $\mathcal{A}_i^\dagger$ where $a=i?^{m,w,c}j$ or $a\in\Sigma_i^{int}$ is allowed in $\mathcal{A}_i^\ddagger$;
- a local transition $(q,r)\xrightarrow{i!^{m,w,c}j}_{\dagger,i}(q',r')$ in $\mathcal{A}_i^\dagger$ is allowed in $\mathcal{A}_i^\ddagger$ if $\Pi_{i,j}(r)=(\widetilde{x},(\widetilde{x}_{k,n})_{k,n\in\mathcal{I}\setminus\{j\}\times\Lambda},\widetilde{\zeta})$, $\zeta(i,j,m)=\widetilde{\zeta}(i,j,m)$ and $\delta_{i!^m j}(x,\widetilde{x}_{i,m})=(x',x'')$ for some $x''\in X_{i,j,m}$.

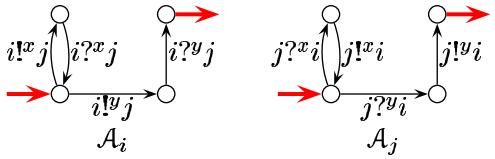


FIG. 5. Safe MPA $\mathcal{S}$

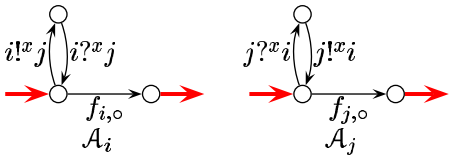


FIG. 6. Realization of $(M^\star)^\dagger$ with $M$ depicted in Fig. 3

This restriction ensures that $\mathcal{A}_i^\ddagger$ is deterministic as opposed to $\mathcal{A}_i$.

Note here that the messages exchanged in the MPAs $\mathcal{S}^\dagger$ and $\mathcal{S}^\ddagger$ are of the form $(m, w, c)$ where $m \in \Lambda$, $w \in X_\mathcal{I}$, and $c \in \Lambda^\dagger$. We claim that the first projection $\gamma : \Lambda \times X_\mathcal{I} \times \Lambda^\dagger \to \Lambda$ yields a bijection from the prefixes of $\mathcal{L}(\mathcal{S}^\ddagger)$ onto the prefixes of $\mathcal{L}$ — i.e. $\mathcal{S}^\ddagger$ simulates $Z$ and $\gamma$ induces an interpretation from $\mathcal{L}(\mathcal{S}^\ddagger)$ onto $\mathcal{L}$. Furthermore $\mathcal{S}^\ddagger$ has no deadlock because $Z$ has no deadlock. This leads us to our first result.

**Theorem 3.7** *Let $\mathcal{L}$ be a regular language of basic MSCs. Then $\mathcal{L}$ is implementable* without deadlock *if and only if $\mathcal{L}$ is coherent.*

## 4. Definitive local stops and spoiling messages

In this section, we introduce two new features that seem to be of some interest in the study of safe implementations with *local* final states, namely definitive local stops and absence of spoiling message. Consider for instance the MPA with two distinct messages $x$ and $y$ depicted on Fig. 5 where local final states are denoted by outgoing bold arrows. This MPA has two interesting features: Whenever each component reaches a final local state, all channels are empty: Any such execution is accepted as a complete behavior. In that way, the continuation of the computation does not depend on the global channel state. Second, final local states are dead states: Whenever a component reaches a final state, it always stops because no transition leaves a final local state. In this case, the continuation of the local computation does not depend on other processes nor on the channel-state.

**Definition 4.1** *Let $\mathcal{S}$ be an MPA with components $(\mathcal{A}_i)_{i\in\mathcal{I}}$ and local final states $(F_i)_{i\in\mathcal{I}}$. We say that*

- *$\mathcal{S}$ has* definitive local stops *if no transition starts from any local final state;*
- *$\mathcal{S}$ has no* spoiling message *if each execution sequence that leads all components to a local final state leads also to the empty channel state.*

We have observed that the MPA depicted in Fig. 5 has definitive local stops and no spoiling message. This MPA is deadlock-free, too.

Consider now some realization with some local final states for the HMSC $M^\star$ with the MSC $M$ of Fig. 3. Such an MPA cannot have definitive local stops: Otherwise the initial state of process $j$ is final and no local transition leaves this state. The same argument holds also if we allow additional message contents. Thus, $M^\star$ is not implementable with definitive local stops, although it is implementable with global final states.

Because MSCs and HMSCs are used as visual objects for early modeling, such specifications usually avoid details concerning internal events. For this reason, we shall relax in the sequel of this paper the notion of interpretation by allowing to add new unspecified local events to the behavior. For instance, Figure 6 shows an MPA with definitive local stops that implements $M^\star$ as soon as we look over the new internal actions $f_{i,\circ}$ and $f_{j,\circ}$. Note that this MPA has one deadlock.

Let us now formalize the relaxed notion of interpretation that we are interested in.

**Definition 4.2** *Let $\Lambda_\circ$ be a new set of messages and $\Sigma_\circ^{int}$ be a new set of internal actions. We denote by $\gamma$ the mapping from MSCs over the set of messages $\Lambda \times \Lambda_\circ$ and the set of internal actions $\Sigma^{int} \uplus \Sigma_\circ^{int}$ to the MSCs over the set of messages $\Lambda$ and the set of internal actions $\Sigma^{int}$ that erases all additional message contents and all internal actions $a \in \Sigma_\circ^{int}$.*

*Let $\mathcal{L}'$ be a set of basic MSCs over $\Lambda \times \Lambda_\circ$ and $\Sigma^{int} \uplus \Sigma_\circ^{int}$. A* proper prefix *of $\mathcal{L}'$ is a prefix of some MSC of $\mathcal{L}'$ such that no maximal event carries an action $a \in \Sigma_\circ^{int}$. Then $\mathcal{L}'$ is a* weak interpretation *of some set $\mathcal{L}$ of basic MSCs over $\Lambda$ and $\Sigma^{int}$ if $\gamma$ induces a bijection from the proper prefixes of $\mathcal{L}'$ onto the prefixes of $\mathcal{L}$.*

We show now how to check whether some given language of basic MSCs can be safely implemented up to some additional message contents and new internal events as an MPA with definitive local stops and no spoiling message.

**Definition 4.3** *Let $f_{i,\circ}$ be some fresh internal action for each $i \in \mathcal{I}$. The* halting language $\mathcal{L}^\dagger$ *of some language of MSCs $\mathcal{L}$ consists of all MSCs of $\mathcal{L}$ extended with one additional internal action $f_{i,\circ}$ at the end of each instance.*

For example, the MPA of Fig. 6 describes the halting language of $M^\star$. This language is not coherent although $M^\star$ is coherent. The next result asserts that $M^\star$ cannot be safely implemented with local final states.

**Theorem 4.4** *Let $\mathcal{L}$ be a regular language of basic MSCs. Then $\mathcal{L}$ admits a weak interpretation $\mathcal{L}'$ which is realizable with local final states, definitive local stops, no spoiling message, nor deadlock if and only if the halting language $\mathcal{L}^\dagger$ is coherent.*

## 5. Checking coherence may be hard

Theorem 3.7 and Theorem 4.4 show that coherence characterizes the regular sets of basic MSCs that admit a safe implementation. In practice, it is easy to check whether a regular language of basic MSCs can be implemented without deadlock as soon as it is described by some deterministic finite automaton that accepts its linear extensions. One simply checks the coherence property at each reachable global state. However, because of some state-explosion problem from HMSCs to linear executions, checking coherence of high-level MSC specifications is hard in general, even for the HMSCs that describe regular languages. We can show the following result.

**Theorem 5.1** *Coherence is undecidable for HMSCs (even for the HMSCs that describe channel-bounded languages). Coherence is EXPSPACE-complete for locally synchronized HMSCs and for globally cooperative HMSCs.*

## 6. Conclusion

By means of the notion of coherence, we have given an algorithm to check whether a regular MSC specification can be safely implemented by some message passing system (Th. 3.7). We have shown how this study applies to the additional requirement that final states are build from local dead states and free from spoiling messages (Th. 4.4). In this paper, we have borrowed from [13] a rather restrictive notion of implementation. For instance, considering again the basic MSC $M$ of Fig. 3, we remark that the MPA of Fig. 5 is *not* an implementation of $M^+$. On the other hand, Th. 4.4 shows that $M^+$ is not safely implementable with local final states. Thus, we believe that less restrictive approaches deserve to be studied, too.

Along this paper, we have shown with several examples and results how time-stamping can help the implementation process. However, it is interesting to compare Theorem 5.1 to the complexity results obtained with the alternative approach followed in [1, 2, 11] where additional message contents are forbidden. Safe realizability is undecidable for HMSCs [11, Th. 2] and EXPSPACE-complete for locally synchronized HMSCs [11, Th. 1] and globally cooperative HMSCs [11, Th. 5]. Thus, both approaches of safe realizability lead to identical complexity bounds.

**Acknowledgments.** We thank B. Caillaud, B. Genest, A. Muscholl, and P.S. Thiagarajan for motivating discussions.

## References

[1] Alur R., Etessami K. and Yannakakis M.: *Inference of message sequence charts*. 22nd Intern. Conf. on Software Engineering, ACM (2000) 304–313

[2] Alur R., Etessami K. and Yannakakis M.: *Realizability and verification of MSC graphs*. ICALP 2001, LNCS **2076** (2001) 797–808

[3] Alur R. and Yannakakis M.: *Model Checking of Message Sequence Charts*. CONCUR'99, LNCS **1664** (1999) 114–129

[4] Caillaud B., Darondeau Ph., Hélouët L. and Lesventes G.: *HMSCs as partial specifications... with PNs as completions*. LNCS **2067** (2001) 87–103

[5] Chandy K. M. and Lamport L.: *Distributed Snapshots: Determining Global States of Distributed Systems*. ACM Trans. on Comp. Syst. **1** (1985) 63–75

[6] Genest B., Muscholl A., Seidl H., and Zeitoun M.: *Infinite-state high-level MSCs: Model-checking and Realizability*. LNCS **2308** (2002) 657–668

[7] Hélouët L. and Jard C.: *Conditions for synthesis of communicating automata from HMSCs*. 5th Intern. Workshop on Formal Methods for Industrial Critical Systems, GMD Report No. **91** (2000) 203–224

[8] Henriksen J.G, Mukund M., Narayan Kumar, K. and Thiagarajan P.S.: *Towards a theory of regular MSC languages*. Tech. rep. (BRICS RS-99-52, 1999)

[9] Henriksen J.G., Mukund M., Narayan Kumar K. and Thiagarajan P.S.: *On message sequence graphs and finitely generated regular MSC language*. ICALP 2000, LNCS **1853** (2000) 675–686

[10] Lamport L.: *Time, Clocks and the Ordering of Events in a Distributed System*. Comm. of the ACM, vol. 21, n **27** (1978) 558–565

[11] Lohrey M.: *Safe Realizability of High-Level Message Sequence Charts*. CONCUR 2002, LNCS **2421** (2002) 177–193

[12] Mattern F.: *Virtual Time and Global States of Distributed Systems*. Proc. Workshop on Parallel and Distributed Algorithms, Elsevier (1989) 215–226

[13] Mukund M., Narayan Kumar K. and Sohoni M.: *Synthesizing distributed finite-state systems from MSCs*. CONCUR 2000, LNCS **1877** (2000) 521–535

[14] Mukund M., Narayan Kumar K. and Sohoni M.: *Bounded time-stamping in message-passing systems*. Theoretical Computer Science **290** (2003) 221–239

[15] Muscholl A. and Peled D.: *Message sequence graphs and decision problems on Mazurkiewicz traces*. MFCS'99, LNCS **1672** (1999) 81–91

[16] Stefanescu A., Esparza J., and Muscholl A.: *Synthesis of distributed algorithms using asynchronous automata*. CONCUR 2003, LNCS (2003) 27–41

[17] Zielonka W.: *Notes on finite asynchronous automata*. R.A.I.R.O. — Inf. Théor. et Appl. **21** (1987) 99–135