

# Synthesis of Safe Message-Passing Systems\*

Nicolas BAUDRU & Rémi MORIN

Aix-Marseille universités — UMR 6166 — CNRS  
Laboratoire d'Informatique Fondamentale de Marseille  
163, avenue de Luminy, F-13288 Marseille Cedex 9, France

**Abstract.** We show that any regular set of basic MSCs can be implemented by a *deadlock-free* communicating finite-state machine with *local termination*: Processes stop in local dead-states independently from the contents of channels and the local states of other processes. We present a self-contained, direct, and relatively simple construction based on a new notion called context MSC.

## Introduction

Message Sequence Charts (MSCs) are a popular model often used for the documentation of telecommunication protocols. They profit by a standardized visual and textual presentation and are related to other formalisms such as sequence diagrams of UML. An MSC gives a graphical description of communications between processes. It usually abstracts away from the values of variables and the actual content of messages. Such specifications are implicitly subjected to some refinement before implementation.

The class of *regular* sets of MSCs introduced in [10] is of particular interest. These languages can be described by finite automata because the number of messages within channels is bounded. Regular languages enjoy several other logical and algebraic properties and they can be model-checked with the help of specific techniques (see e.g. [1]). The theory of regular MSC languages has been extended in various directions [3, 7–9]. In particular [3] and [7] extend to the framework of unbounded channels one of the main result from [10]: *Any regular set of MSCs can be implemented by a communicating finite-state machine (for short, a CFM) with bounded channel capacities.*

Yet, the main drawback of the CFMs built in [3, 7, 10] is that they possibly lead to deadlocks. In this paper we improve that result and prove that we can make sure that the CFM built from a regular set of MSCs is *deadlock-free*. As opposed to [3, 7, 10] the CFMs we consider satisfy two other interesting properties. First, processes stop in local final states independently from the local states of other processes, that is, we adopt a *local acceptance condition* similarly to [1, 8]. Second, *final local states are dead-states*: Differently from [1, 8] we require that no process can leave any final local state, that is, each process *terminates locally*. This second requirement is particularly relevant because deadlock-free CFMs with local termination are *stuck-free*: Whenever all processes stop, no unexpected message remains within the channels. This is the main difference with [1, 3, 7, 8, 10] for which the acceptance condition ensures that all channels are empty: The system relies implicitly on a global supervisor that checks emptiness of all channels and controls the termination of all processes. In this paper

---

\* Supported by the ANR project SOAPDC

we do not assume any global supervisor. We build CFMs *with local termination* that are *deadlock-free* and such that *any accepting execution leads to empty channels*. The necessary counterpart of these strong requirements is that we build *non-deterministic* CFMs. In particular CFMs may have multiple (finitely many) initial global states: Intuitively this means that processes can synchronize in a preliminary phase in order to agree on some decisions before the system starts. Similarly to [3, 7, 10] and differently from [1, 8] the implementation process allows to add information to specified messages. This refinement implements intuitively a kind of *distributed control*.

A proof sketch of our result relies on the rich theory of Mazurkiewicz traces [5] and proceeds as follows. A first step due to Kuske [11] encodes the given regular set of MSCs  $L$  into a regular trace language  $L'$  over some independence alphabet that depends on the channel-bound of  $L$ . Next one applies directly a variant of Zielonka's theorem [13] which asserts that  $L'$  is accepted by a deadlock-free non-deterministic asynchronous cellular automaton  $Z$  with local termination. It remains then only to turn  $Z$  into a deadlock-free CFM with local termination that accepts  $L$ . As opposed to Kuske's encoding, *this last step is unfortunately not easy*. The main reason is that components of asynchronous cellular automata synchronize by means of shared variables whereas processes of a CFM exchange messages. In [2] we designed a rather involved method based on a bounded time-stamp protocol by Mukund, Narayan Kumar and Sohoni [12] in order to build a deadlock-free deterministic CFM from a deadlock-free deterministic asynchronous automaton. This approach can be adapted in order to preserve local termination and yield the expected deadlock-free and stuck-free CFM.

Let us now explain why we choose not to develop this proof sketch. First the technique from [2] is particularly suitable for deterministic CFMs but it is rather complicated. Since we consider here non-deterministic CFMs, we prefer to present a *simpler* construction that consists of a single technical lemma and two basic inductions. Second we believe that our *direct and self-contained approach* is more valuable than referring to the analogous result in the setting of Mazurkiewicz traces [13]. Finally there are only few known methods to build CFMs from regular languages so our *the new inductive approach* may be also interesting by itself.

This paper is organized as follows. We introduce in Section 1 a straightforward and natural extension of basic MSCs called *context MSCs*: The latter are simply compositional MSCs [9] provided with a channel-state. Context MSCs come equipped with some associative product which is useful to decompose regular languages of basic MSCs into simpler components inductively in a Kleene-like manner. From an algebraic viewpoint, the resulting structure turns out to be a concurrency monoid [6] in which basic MSCs form a submonoid. In Section 2 we formalize the model of CFMs with local termination together with the key notion of a deadlock. As announced above, deadlock-free CFMs with local termination are stuck-free. Then we present our technical result (Section 3) and finally conclude by means of two elementary decomposition techniques (Section 4).

## 1 Message sequence charts

Following a classical trend of concurrency theory the executions of a distributed system are regarded as labeled partial orders (called pomsets). Although our result holds for non-FIFO channels we assume in this paper that all channels are FIFO in order to simplify the presentation. Furthermore, for the same reason, the actual content of messages are abstracted from the notion of MSCs similarly to the approach adopted in [3, 7, 10].

In this paper, we call alphabet any non-empty set; elements of alphabets are called *actions*. A *pomset* over an alphabet  $\Sigma$  is a triple  $t = (E, \preceq, \xi)$  where  $(E, \preceq)$  is a finite partial order and  $\xi$  is a mapping from  $E$  to  $\Sigma$  *without autoconcurrency*:  $\xi(x) = \xi(y)$  implies  $x \preceq y$  or  $y \preceq x$  for all  $x, y \in E$ . A pomset can be seen as an abstraction of 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  $e \in E$ . Furthermore, the order  $\preceq$  describes the causal dependence between events. Let  $t = (E, \preceq, \xi)$  be a pomset and  $x, y \in E$ . Then  $y$  *covers*  $x$  (denoted  $x \prec y$ ) if  $x \prec y$  and  $x \prec z \preceq y$  implies  $y = z$ . An event  $x$  is *minimal* if  $y \preceq x$  implies  $y = x$ .

An *order extension* of a pomset  $t = (E, \preceq, \xi)$  is a pomset  $t' = (E, \preceq', \xi)$  such that  $\preceq \subseteq \preceq'$ . A *linear extension* of  $t$  is an order extension that is linearly ordered. It corresponds to a sequential view of the concurrent execution  $t$ . Linear extensions of a pomset  $t$  over  $\Sigma$  can naturally be regarded as words over  $\Sigma$ . By  $\text{LE}(t) \subseteq \Sigma^*$ , we denote the set of linear extensions of a pomset  $t$  over  $\Sigma$ .

An *ideal* of a pomset  $t = (E, \preceq, \xi)$  is a downward-closed subset  $H \subseteq E$ :  $x \in H \wedge y \preceq x \Rightarrow y \in H$ . The restriction  $t' = (H, \preceq \cap (H \times H), \xi \cap (H \times \Sigma))$  is called a *prefix* of  $t$  and we write  $t' \leq t$ . For all  $z \in E$ , we denote by  $\downarrow_t z$  the ideal of events below  $z$ , i.e.  $\downarrow_t z = \{y \in E \mid y \preceq z\}$ . We denote by  $|t|_a$  the number of events  $x \in E$  such that  $\xi(x) = a$ .

### 1.1 Basic and context message sequence charts

Message sequence charts are defined in the Z.120 recommendation of the ITU-T with a formal syntax and graphical rules. They can be seen also as particular pomsets over some alphabet that we introduce first. Let  $\mathcal{I}$  be a finite set of processes (also called *instances*). For any instance  $i \in \mathcal{I}$ , the alphabet  $\Sigma_i$  is the disjoint union of the set of *send actions*  $\Sigma_i^! = \{i!j \mid j \in \mathcal{I} \setminus \{i\}\}$  and the set of *receive actions*  $\Sigma_i^? = \{i?j \mid j \in \mathcal{I} \setminus \{i\}\}$ . Observe that the alphabets  $\Sigma_i$  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  $\text{Ins}(a)$  the unique instance  $i$  such that  $a \in \Sigma_i$ , that is the particular instance on which each occurrence of action  $a$  occurs. Finally, for any pomset  $(E, \preceq, \xi)$  over  $\Sigma_{\mathcal{I}}$  we denote by  $\text{Ins}(e)$  the instance on which the event  $e \in E$  occurs:  $\text{Ins}(e) = \text{Ins}(\xi(e))$ .

A channel-state describes the number of messages in transit at some stage of an execution. Formally we let  $\mathcal{K} = \{(i, j) \in \mathcal{I} \times \mathcal{I} \mid i \neq j\}$  denote the set of all channels within the instances  $\mathcal{I}$ . Then a channel-state is simply a mapping  $\chi : \mathcal{K} \rightarrow \mathbb{N}$ . The *empty channel-state*  $\bar{0}$  maps each channel to 0. Let  $\chi$  be a channel-state and  $M = (E, \preceq, \xi)$  be a pomset over  $\Sigma_{\mathcal{I}}$ . We say that two events  $e, f \in E$  match each other w.r.t.  $\chi$  if  $e$  is a send event from  $i$  to  $j$  and  $f$  is the corresponding receive event on  $j$ : Formally, we put  $e \sim_{\chi} f$  if  $\xi(e) = i!j$ ,  $\xi(f) = j?i$ , and moreover  $\chi(i, j) + \downarrow_M e|_{i!j} = \downarrow_M f|_{j?i}$ .

DEFINITION 1.1. A context MSC is a pair  $(M, \chi)$  where  $M = (E, \preceq, \xi)$  is a pomset over  $\Sigma_{\mathcal{I}}$  and  $\chi \in \mathbb{N}^{\mathcal{K}}$  is a channel-state such that

$$\mathbf{M}_1: \forall e, f \in E: \text{Ins}(e) = \text{Ins}(f) \Rightarrow (e \preceq f \vee f \preceq e)$$

$$\mathbf{M}_2: \forall e, f \in E: e \rightsquigarrow_{\chi} f \Rightarrow e \preceq f$$

$$\mathbf{M}_3: \forall e, f \in E: [e \not\prec f \wedge \text{Ins}(e) \neq \text{Ins}(f)] \Rightarrow e \rightsquigarrow_{\chi} f$$

$$\mathbf{M}_4: \forall (i, j) \in \mathcal{K}: \chi(i, j) + |M|_{i!j} \geq |M|_{j?i}$$

A context MSC  $(M, \chi)$  is also denoted by  $M@_{\chi}$ . By  $\mathbf{M}_1$ , events occurring on the same instance are linearly ordered: Non-deterministic choice cannot be described within an MSC. Axiom  $\mathbf{M}_2$  formalizes that the reception of any message will occur after the corresponding send event. By  $\mathbf{M}_3$ , causality in  $M$  consists only in the linear dependency over each instance and the ordering of pairs of corresponding send and receive events.

Let  $M@_{\chi}$  be a context MSC. Then  $\chi$  is called the *domain* of  $M@_{\chi}$ . The *codomain* of  $M@_{\chi}$  is the channel-state  $\chi'$  such that  $\chi'(i, j) = \chi(i, j) + |M|_{i!j} - |M|_{j?i}$  for all channels  $(i, j) \in \mathcal{K}$ . Axiom  $\mathbf{M}_4$  ensures that the codomain of a context MSC is a channel-state. It is clear that *the usual set of basic MSCs can be identified with the subset of context MSCs whose domain and codomain are the empty channel-state*. Observe here that context MSCs satisfy the following *consistence property*: If two context MSCs share the same domain and a common linear extension then they are identical.

## 1.2 Semigroup of context message sequence charts

We come now to the definition of the concatenation of two context MSCs. First we add formally a special context MSC  $0$  to the set of context MSCs. This additional context MSC  $0$  is called *non-valid* and will act as a zero: We put  $x \cdot 0 = 0 \cdot x = 0$ .

DEFINITION 1.2. Let  $M_1@_{\chi_1} = (E_1, \preceq_1, \xi_1, \chi_1)$  and  $M_2@_{\chi_2} = (E_2, \preceq_2, \xi_2, \chi_2)$  be two valid MSCs. Let  $\rightsquigarrow$  be the binary relation over  $E_1 \times E_2$  such that  $e_1 \rightsquigarrow e_2$  if  $\xi_1(e_1) = i!j$ ,  $\xi_2(e_2) = j?i$ , and  $\chi_1(i, j) + |\downarrow_{M_1} e_1|_{i!j} = |M_1|_{j?i} + |\downarrow_{M_2} e_2|_{j?i}$ .

If the codomain of  $M_1@_{\chi_1}$  is  $\chi_2$  then the product  $M_1@_{\chi_1} \cdot M_2@_{\chi_2}$  is the context MSC  $(E, \preceq, \xi, \chi_1)$  where  $E = E_1 \uplus E_2$ ,  $\xi = \xi_1 \cup \xi_2$  and the partial order  $\preceq$  is the transitive closure of  $\preceq_1 \cup \preceq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \text{Ins}(e_1) = \text{Ins}(e_2)\} \cup \rightsquigarrow$ .

If the codomain of  $M_1@_{\chi_1}$  is not  $\chi_2$  then  $M_1@_{\chi_1} \cdot M_2@_{\chi_2} = 0$ .

This product extends the usual concatenation of basic MSCs viewed as the subset of context MSCs whose domain and codomain are the empty channel-state  $\bar{0}$ . The consistence property allows us to characterize this product as follows.

PROPOSITION 1.3. Let  $M_1@_{\chi_1}$  and  $M_2@_{\chi_2}$  be two valid context MSCs such that the codomain of  $M_1@_{\chi_1}$  is  $\chi_2$ . Let  $u_1$  and  $u_2$  be some linear extensions of  $M_1$  and  $M_2$  respectively. Then the product  $M_1@_{\chi_1} \cdot M_2@_{\chi_2}$  is the valid context MSC  $M@_{\chi_1}$  such that  $u_1 \cdot u_2 \in \text{LE}(M)$ .

Let  $\text{cMSC}$  denote the set of all (valid and non-valid) context MSCs. Proposition 1.3 above enables us to check easily that the product of context MSCs is associative. Thus the set of context MSCs forms a semigroup. The proof of our main result relies on a representation of MSC languages in the form of rational expressions built by means of unions  $(\mathcal{L}_1 + \mathcal{L}_2)$ , products  $(\mathcal{L}_1 \cdot \mathcal{L}_2)$ , and strict iterations  $(\mathcal{L}^+ = \bigcup_{k \geq 1} \mathcal{L}^k)$ . We could identify formally all empty context MSCs as a single context MSC and get a concurrency monoid [4, 6].

### 1.3 Regular sets of MSCs

Let  $\chi_1, \chi_2$  be two channel-states. A subset of valid context MSCs  $\mathcal{L} \subseteq \text{cMSC} \setminus \{0\}$  is *located* at  $(\chi_1, \chi_2)$  if all context MSCs from  $\mathcal{L}$  have domain  $\chi_1$  and codomain  $\chi_2$ . Then  $\chi_1$  and  $\chi_2$  are called respectively the domain and the codomain of  $\mathcal{L}$ .

DEFINITION 1.4. *A located set of MSCs  $\mathcal{L}$  is regular if the corresponding set of words  $\text{LE}(\mathcal{L}) = \bigcup_{M \in \mathcal{L}} \text{LE}(M)$  is recognizable in the free monoid  $\Sigma_{\mathcal{T}}^*$ . A set of context MSCs is regular if it is a finite union of regular located sets of context MSCs.*

In particular a subset of basic MSCs is regular in the sense of [10] if and only if it is regular according to the above definition.

For later purposes, we need to extend the usual notion of channel-bounded languages from basic MSCs to context MSCs as follows. The *channel-width* of a valid context MSC  $M @ \chi$  is

$$\max_{u \in \text{LE}(M)} \max_{v \leq u} \max_{(i,j) \in \mathcal{K}} \chi(i,j) + |v|_{i!j} - |v|_{j?i}.$$

Intuitively the channel-width of  $M @ \chi$  is the maximal number of messages that may be in transit within some channel at any stage of the execution of  $M @ \chi$ . A subset of valid context MSCs  $\mathcal{L}$  is *channel-bounded* by  $B \in \mathbb{N}$  if each context MSC from  $\mathcal{L}$  has a channel-width at most  $B$ .

Consider now a regular set of context MSCs  $\mathcal{L}$  located at  $(\chi_1, \chi_2)$  and the minimal deterministic automaton  $\mathcal{A} = (Q, \iota, F, \longrightarrow)$  over  $\Sigma_{\mathcal{T}}$  that accepts  $\text{LE}(\mathcal{L})$ . All states of  $\mathcal{A}$  are reachable from the initial state  $\iota \in Q$  and co-reachable from the subset of final states  $F \subseteq Q$ . The next basic observation asserts that each state from  $\mathcal{A}$  corresponds to some particular channel-state.

PROPOSITION 1.5. *There exists a mapping  $\chi : Q \rightarrow \mathbb{N}^{\mathcal{K}}$  such that  $\chi(\iota) = \chi_1$ ,  $\chi(q) = \chi_2$  for all  $q \in F$ , and  $q \xrightarrow{u} q'$  implies  $\chi(q')(i,j) = \chi(q)(i,j) + |u|_{i!j} - |u|_{j?i}$  for all  $q, q' \in Q$  and all channels  $(i,j) \in \mathcal{K}$ .*

It follows that any regular set of context MSCs is channel-bounded.

## 2 Deadlock-free and stuck-free message-passing systems

In this section we introduce the model of communicating finite-state machines and the related notions of deadlock, local termination, and stuck messages. The semantics of these systems is given in a natural way by means of sets of MSCs.

### 2.1 Communicating finite-state machines with local termination

Recall here that MSC specifications are used usually at an early stage of the design so that a refinement procedure can occur before implementation. In this paper refinement corresponds to the possibility to add some control information to messages in order to be able to build a correct implementation. To do so we use a fixed set  $\Lambda$  of control messages that will be added to the contents of specified messages. We denote by  $i!^m j$  the action by  $i$  that sends a message with control information  $m$  to  $j$ . Its receipt by  $j$  is denoted by  $j?^m i$ . We put  $\Sigma_i^{\Lambda} = \{i!^m j, j?^m i \mid j \in \mathcal{I} \setminus \{i\}, m \in \Lambda\}$  and  $\Sigma_{\mathcal{T}}^{\Lambda} = \bigcup_{i \in \mathcal{I}} \Sigma_i^{\Lambda}$ .

A *refined channel-state* describes the sequence of control informations associated with the sequence of messages in transit; it is formalized as a map  $\rho : \mathcal{K} \rightarrow \Lambda^*$ .

A *communicating finite-state machine* (for short, a *CFM*) over  $\Lambda$  consists of a process  $\mathcal{A}_i = (Q_i, \longrightarrow_i, F_i)$  for each instance  $i \in \mathcal{I}$  together with a *finite* set of initial global states  $I \subseteq (\prod_{i \in \mathcal{I}} Q_i) \times (\Lambda^*)^{\mathcal{K}}$  where  $Q_i$  is a finite set of local states for process  $i$ ,  $\longrightarrow_i \subseteq Q_i \times \Sigma_i^A \times Q_i$  is a local transition relation for  $i$ , and  $F_i \subseteq Q_i$  is a subset of final local states. All along this paper we require additionally that *all final local states are dead*: For all instances  $i$  and for all final local states  $q_i \in F_i$ , there is no transition  $q_i \xrightarrow{a} q'_i$  for all  $a \in \Sigma_i \Lambda$  and all  $q'_i \in Q_i$ . Thus we consider only CFMs with *local termination*.

In this setting, a *global state* is a pair  $s = (q, \rho)$  where  $q \in \prod_{i \in \mathcal{I}} Q_i$  is a tuple of local states and  $\rho : \mathcal{K} \rightarrow \Lambda^*$  is a refined channel-state. For all global states  $s = (q, \rho)$  with  $q = (q_i)_{i \in \mathcal{I}}$  and all  $i \in \mathcal{I}$  we put  $s \downarrow i = q_i$ . A global state  $s$  is *final* if  $s \downarrow i \in F_i$  for all  $i \in \mathcal{I}$ . Thus  $F = (\prod_{i \in \mathcal{I}} F_i) \times (\Lambda^*)^{\mathcal{K}}$  denotes the set of all final global states.

Intuitively each process stops independently from the current contents of channels and independently from the local states of other processes. This approach is somehow more restrictive than [1, 3, 7, 8, 10] which assume that final global states are associated with the empty channel-state. On the other hand we allow multiple (finitely many) initial global states and consequently we consider in this paper *non-deterministic* CFMs.

## 2.2 Deadlocks and stuck messages

The *system of global states* associated to a communicating finite-state machine  $\mathcal{S}$  is the transition system  $\mathcal{A}_{\mathcal{S}} = (S, \longrightarrow)$  where  $S = \prod_{i \in \mathcal{I}} Q_i \times (\Lambda^*)^{\mathcal{K}}$  is the set of all global states and the global transition relation  $\longrightarrow \subseteq S \times \Sigma_{\mathcal{I}}^A \times S$  satisfies the two next properties for any global states  $s = (q, \rho)$  and  $s' = (q', \rho')$ :

- for all distinct instances  $i$  and  $j$ ,  $s \xrightarrow{i!^m j} s'$  if
  1.  $s \downarrow i \xrightarrow{i!^m j} s' \downarrow i$  and  $s' \downarrow k = s \downarrow k$  for all  $k \in \mathcal{I} \setminus \{i\}$ ,
  2.  $\rho'(i, j) = \rho(i, j) \cdot m$  and  $\rho(x) = \rho'(x)$  for all  $x \in \mathcal{K} \setminus \{(i, j)\}$ ;
- for all distinct instances  $i$  and  $j$ ,  $s \xrightarrow{j?^m i} s'$  if
  1.  $s \downarrow j \xrightarrow{j?^m i} s' \downarrow j$  and  $s' \downarrow k = s \downarrow k$  for all  $k \in \mathcal{I} \setminus \{j\}$ ,
  2.  $\rho(i, j) = m \cdot \rho'(i, j)$  and  $\rho(x) = \rho'(x)$  for all  $x \in \mathcal{K} \setminus \{(i, j)\}$ .

As usual with transition systems, for any word  $u = a_1 \dots a_n$  over  $\Sigma_{\mathcal{I}}^A$ , we write  $s \xrightarrow{u} s'$  if there are some global states  $s_0, \dots, s_n \in S$  such that  $s_0 = s$ ,  $s_n = s'$  and for each  $r \in [1, n]$ ,  $s_{r-1} \xrightarrow{a_r} s_r$ . For all global states  $s_1, s_2 \in S$  we denote by  $L(\mathcal{S}, s_1, s_2)$  the set of words  $u$  over  $\Sigma_{\mathcal{I}}^A$  such that  $s_1 \xrightarrow{u} s_2$ . We say that a CFM  $\mathcal{S}$  is *safe* if all global states reachable from  $I$  are co-reachable from  $F$ . In other words, a safe CFM has no *deadlock*.

Let  $\pi : \Sigma_{\mathcal{I}}^A \rightarrow \Sigma_{\mathcal{I}}$  be the mapping that forgets the additional control information:  $\pi(i!^m j) = i!j$  and  $\pi(j?^m i) = j?i$ . This mapping extends in the obvious way to a map from words over  $\Sigma_{\mathcal{I}}^A$  to words over  $\Sigma_{\mathcal{I}}$ . For any refined channel-state  $\rho$ ,  $\pi(\rho)$  denotes the channel-state  $\chi$  such that  $\chi(i, j)$  is the length of  $\rho(i, j)$  for all  $(i, j) \in \mathcal{K}$ .

Consider now a CFM  $\mathcal{S}$  and two global states  $s_1, s_2$  with respective refined channel-states  $\rho_1, \rho_2$ . For any word  $u \in L(\mathcal{S}, s_1, s_2)$  there exists a unique context MSC  $M @ \chi$

such that  $\chi = \pi(\rho_1)$  and  $\pi(u)$  is a linear extension of  $M$ . Moreover  $M@_\chi$  has codomain  $\pi(\rho_2)$ . The *language of context MSCs*  $\mathcal{L}(\mathcal{S})$  *accepted* by  $\mathcal{S}$  consists of all valid context MSCs  $M@_\chi$  such that there are two global states  $s = (q, \rho) \in I$  and  $s' = (q', \rho') \in F$  with  $\pi(\rho) = \chi$  and a word  $v \in \text{LE}(M)$  such that  $v \in \pi(L(\mathcal{S}, s, s'))$ . Noteworthy, it can be easily shown that this condition ensures that *all* linear extensions of  $M$  belong to  $\pi(L(\mathcal{S}, s, s'))$ . Observe also that if  $\mathcal{L}(\mathcal{S})$  consists of *basic* MSCs and  $\mathcal{S}$  is *safe* then all initial global states and all *reachable* final global states are associated with the empty channel-state: Thus there are no message stuck in channels when the system stops.

### 2.3 Implementable languages: Two basic properties

We say that a language  $\mathcal{L}$  of context MSCs is *implementable* if there exists a *safe* CFM that accepts  $\mathcal{L}$ . Clearly any finite union of implementable languages is implementable. Observe now that for any implementable *located* set  $\mathcal{L}$  of context MSCs, there exists a safe CFM that accepts  $\mathcal{L}$  and such that *all initial global states*  $s = (q, \rho)$  *share a common refined channel-state*  $\rho$ . Now it is not difficult to check that the product of two implementable located languages is implementable if this product is valid.

LEMMA 2.1. *Let  $\mathcal{L}_1$  and  $\mathcal{L}_2$  be two implementable sets of context MSCs.*

1.  $\mathcal{L}_1 + \mathcal{L}_2$  *is implementable.*
2. *If  $\mathcal{L}_1 \cdot \mathcal{L}_2$  is valid, i.e.  $0 \notin \mathcal{L}_1 \cdot \mathcal{L}_2$ , then  $\mathcal{L}_1 \cdot \mathcal{L}_2$  is implementable.*

## 3 Iteration of implementable languages

In this section we establish for the iteration operation a result analogous to Lemma 2.1. With no surprise dealing with iteration turns out to be more complicated.

Let  $k \in \mathcal{I}$  be some fixed instance. A context MSC  $M@_\chi$  is *initiated* (by  $k$ ) if  $M$  admits a least event and this event is labeled by some send action from  $k$ . A *located* set of context MSCs  $\mathcal{L}$  is *initiated* (by  $k$ ) if all context MSCs from  $\mathcal{L}$  are initiated (by  $k$ ).

THEOREM 3.1. *Let  $\mathcal{L}$  be some initiated and implementable set of context MSCs located at some  $(\chi_0, \chi_0)$ . If  $\mathcal{L}^+$  is channel-bounded then  $\mathcal{L}^+$  is implementable, too.*

This section is devoted to the proof of this result. We fix some some initiated and implementable set of context MSCs  $\mathcal{L}$  located at some  $(\chi_0, \chi_0)$ . We assume that  $\mathcal{L}^+$  is channel-bounded by  $B$ . Let  $\mathcal{S}$  be a safe CFM over  $\Lambda$  that accepts  $\mathcal{L}$ . We denote by  $\mathcal{A}_i$  the local process of instance  $i$  in  $\mathcal{S}$ . We can assume that messages initially in channels do not carry any relevant control information, that is, we assume formally that for all initial global state  $s = (q, \rho)$ , any global state  $\tilde{s} = (q, \tilde{\rho})$  with  $\pi(\tilde{\rho}) = \chi_0$  is initial, too.

### 3.1 Intuitive description of the consensus protocol

We build from  $\mathcal{S}$  a safe CFM  $\mathcal{S}'$  that accepts  $\mathcal{L}^+$ . Control messages exchanged within  $\mathcal{S}'$  are pairs  $(m, \tau)$  where  $m \in \Lambda$  is a control message from  $\mathcal{S}$  and  $\tau$  is a tag added by  $\mathcal{S}'$ . Process  $k$  will act as a leader within  $\mathcal{S}'$ : It will make some choices along the executions and these choices will be formalized and communicated to other processes by means

of these tags. The choices made by  $k$  and the tags used by  $S'$  are essentially built upon the subset  $I$  of initial global states of  $S$ . We say that an instance  $i$  is *live* in some initial global state  $s \in I$  if the local state  $s \downarrow i$  is not final for the local process  $A_i$  of  $S$ : We put  $\text{Live}(s) = \{i \in \mathcal{I} \mid s \downarrow i \notin F_i\}$ . Since  $\mathcal{L}$  is initiated,  $k \in \text{Live}(s)$  for all  $s \in I$ .

Basically each process  $A'_i$  of  $S'$  simulates and iterates the behaviors of  $A_i$ : It possibly starts a new execution when it reaches a final local state of  $A_i$ . However the global behaviors of the whole system  $S'$  must correspond also to iterations of  $\mathcal{L}$ : Any execution of  $S'$  have to appear as a sequence of *phases* that simulate each an execution of  $S$ . That is why all processes must follow a *consensus protocol* that determines at each step which processes should take part in the next phase and from which local states they should start. Since  $\mathcal{L}$  is initiated by  $k$ , all other instances start any execution from  $S$  by receiving a first message, called the *initiating message*. The tag added to this message by  $S'$  specifies from which local state of  $S$  each instance should start a new phase.

The first role of process  $k$  is to choose on-the-fly a sequence of initial global states  $s_1, \dots, s_n \in I$  from  $S$  and initiate a new simulation of some execution of  $S$  from  $s_m$  as soon as it has finished the previous phase from  $s_{m-1}$ . In doing so, it moves from a final local state of  $A_k$  to the local state of  $A_k$  that corresponds to  $s_m$  and sends its first message with a tag that includes  $s_m$ . These actions are considered atomic. Instances that are not live in  $s_m$  will not take part in this phase.

The second role of process  $k$  is to choose on-the-fly a subset of processes that must terminate—that is, that will not take part in further phases. This information is necessary because each process has to know when it does not need to wait any longer for a new initiating message, that is, when it reaches a final local state of  $S'$ . The choice of terminating instances is included in the tag of messages exchanged by  $S'$  within a phase. Thus process  $k$  keeps track of the subset  $\mathcal{H}$  of instances that have already terminated in previous phases. Obviously the subset  $\mathcal{H} \subseteq \mathcal{I}$  grows from phase to phase. In order to avoid deadlocks, process  $k$  makes sure that the next phase can be achieved by non-terminated processes, that is, the next phase starts from some  $s \in I$  with  $\text{Live}(s) \subseteq \mathcal{I} \setminus \mathcal{H}$ . Moreover process  $k$  chooses among the live instances of  $s$  the subset of instances  $\mathcal{G} \subseteq \text{Live}(s)$  that will simulate their last execution of  $S$ . As a consequence the new value of  $\mathcal{H}$  is  $\mathcal{H} \cup \mathcal{G}$ . In that way the sequence of phases  $s_1, \dots, s_n \in I$  is associated with an increasing sequence of dead instances  $\mathcal{H}_1 \subseteq \dots \subseteq \mathcal{H}_n \subseteq \mathcal{I}$ . Since all processes must stop at some point, the choices by process  $k$  must lead eventually to  $\mathcal{H}_n = \mathcal{I}$ .

We detail now how process  $k$  chooses the sequence of initial global states  $s_1, \dots, s_n \in I$  together with the sequence of terminating instances  $\mathcal{H}_1, \dots, \mathcal{H}_n = \mathcal{I}$  starting from some set of initially dead or terminating instances  $\mathcal{H}_0$ . As explained above the sequence  $\mathcal{H}_0, \mathcal{H}_1, \dots, \mathcal{H}_n$  is increasing,  $\mathcal{H}_m \setminus \mathcal{H}_{m-1} \subseteq \text{Live}(s_m)$ ,  $\text{Live}(s_m) \subseteq \mathcal{I} \setminus \mathcal{H}_{m-1}$  and  $\mathcal{H}_n = \mathcal{I}$ . Let us consider the finite directed graph  $\mathfrak{G}$  whose nodes are the subsets  $\mathcal{H}$  of  $\mathcal{I}$  and such that there is an edge from  $\mathcal{H}$  to  $\mathcal{H}'$  if there exists some initial global state  $s \in I$  such that  $\mathcal{H} \subseteq \mathcal{H}'$ ,  $\mathcal{H}' \setminus \mathcal{H} \subseteq \text{Live}(s)$  and  $\text{Live}(s) \subseteq \mathcal{I} \setminus \mathcal{H}$ . A node  $\mathcal{H} \subseteq \mathcal{I}$  is *secure* if there exists a path in  $\mathfrak{G}$  from  $\mathcal{H}$  to  $\mathcal{I}$ . In particular  $\mathcal{I}$  is secure. A pair  $(s, \mathcal{H}') \in I \times 2^{\mathcal{I}}$  is a *secure choice* for  $\mathcal{H}$  if  $\mathcal{H} \subseteq \mathcal{H}'$ ,  $\mathcal{H}' \setminus \mathcal{H} \subseteq \text{Live}(s)$ ,  $\text{Live}(s) \subseteq \mathcal{I} \setminus \mathcal{H}$  and  $\mathcal{H}'$  is secure. Clearly if  $\mathcal{H}$  is secure and  $\mathcal{H} \neq \mathcal{I}$  then there are some secure choices  $(s, \mathcal{H}')$  for  $\mathcal{H}$ . Before starting a new phase, process  $k$  selects arbitrarily a secure choice

$(s, \mathcal{H}')$  and initiates a new phase accordingly. This new phase is associated with the extended set of dead or terminating instances  $\mathcal{H}'$ .

Intuitively all messages exchanged within a phase are tagged with the same information. The tag of a phase consists basically of

- the global initial state  $s \in I$  so that each process  $i \in \mathcal{I}$  knows from which local state  $s \downarrow i$  it should start over, and
- the subset  $\mathcal{H}'$  of instances that will not take part in further phases.

However if the domain  $\chi_0$  of  $\mathcal{L}$  is not empty then each process of  $\mathcal{S}$  consumes a fixed sequence of messages before it receives messages sent within  $\mathcal{S}$ . Therefore each process  $\mathcal{A}'_i$  receives first from  $j$  a fixed sequence of messages with a tag possibly different from the ongoing phase and accepts only messages with some correct tag afterwards.

Now it is crucial that two concurrent phases associated with the same tag do not interfere. For that reason process  $k$  counts the number of phases in which each instance is live modulo some constant  $D$  by means of some counter  $\kappa : \mathcal{I} \rightarrow [0, D - 1]$  and adds this counter to the tag of phases. Thus tags are actually triples  $(s, \mathcal{H}', \kappa)$ . We take  $D = |\mathcal{I}| + B + 1$  where  $|\mathcal{I}|$  is the number of instances and  $\mathcal{L}^+$  is channel-bounded by  $B$ . The proof of our technical lemma below (Lemma 3.3) explains why these counters ensure that phases with the same tag cannot interfere.

### 3.2 Formal construction of $\mathcal{S}'$

We define now formally the processes  $\mathcal{A}'_i$  of the CFM  $\mathcal{S}'$  according to the above intuitions. Let  $T = I \times 2^{\mathcal{I}} \times [0, D - 1]^{\mathcal{I}}$  be the set of all tags. The set of messages used by  $\mathcal{S}'$  is  $\Lambda' = \Lambda \times T$ . A local state of  $\mathcal{A}'_i$  is a triple  $r = (q, \tau, \chi)$  where  $q$  is a local state of  $\mathcal{A}_i$ ,  $\tau$  is a tag, and  $\chi$  is a channel-state bounded by  $B$ . The latter enables each process to ensure that the appropriate number of messages from the past are received along each channel. Let  $i$  be some instance. Let  $r = (q, \tau, \chi)$  and  $r' = (q', \tau', \chi')$  be two local states of  $\mathcal{A}'_i$  where  $\tau = (s, \mathcal{H}, \kappa)$  and  $\tau' = (s', \mathcal{H}', \kappa')$ . We put  $r \xrightarrow{a}_i r'$  in  $\mathcal{A}'_i$  if one of the next conditions is satisfied:

1. Instance  $i$  is  $k$  and it initiates a new phase:  $i = k$ ,  $q \in F_i$ ,  $i \notin \mathcal{H}$ ,  $a = i!^{m, \tau'} j$ ,  $s' \downarrow i \xrightarrow{i!^{m, \tau'} j}_i q'$  in  $\mathcal{A}_i$ ,  $\chi' = \chi_0$ ,  $(s', \mathcal{H}')$  is a secure choice for  $\mathcal{H}$ ,  $\kappa'(l) = \kappa(l) + 1 \pmod{D}$  for all  $l \in \text{Live}(s')$ , and  $\kappa'(l) = \kappa(l)$  for all  $l \notin \text{Live}(s')$ .
2. Instance  $i$  is not  $k$  and it starts a new phase:  $i \neq k$ ,  $q \in F_i$ ,  $i \notin \mathcal{H}$ ,  $a = i?^{m, \tau'} j$ ,  $s' \downarrow i \xrightarrow{i?^{m, \tau'} j}_i q'$  in  $\mathcal{A}_i$ ,  $\chi' = \chi_0$ ,  $\chi_0(j, i) = 0$ ,  $\mathcal{H} \subseteq \mathcal{H}'$  and  $\kappa'(i) = \kappa(i) + 1 \pmod{D}$ .
3. Process  $i$  goes on the current phase and receives a message from a previous phase:  $\tau = \tau'$ ,  $a = i?^{n, \tau''} j$ ,  $q \xrightarrow{i?^{n, \tau''} j}_i q'$  in  $\mathcal{A}_i$ ,  $\chi(j, i) \geq 1$ ,  $\chi'(j, i) = \chi(j, i) - 1$  and  $\chi(x) = \chi'(x)$  for all  $x \neq (j, i)$ .
4. Process  $i$  goes on the current phase and receives a message with the current tag:  $\tau = \tau'$ ,  $a = i?^{m, \tau} j$ ,  $q \xrightarrow{i?^{m, \tau} j}_i q'$  in  $\mathcal{A}_i$ ,  $\chi(j, i) = 0$  and  $\chi = \chi'$ .
5. Process  $i$  goes on the current phase and sends a message:  $\tau = \tau'$ ,  $a = i!^{m, \tau} j$ ,  $q \xrightarrow{i!^{m, \tau} j}_i q'$  in  $\mathcal{A}_i$ , and  $\chi = \chi'$ .

A local state  $r = (q, \tau, \chi)$  of  $\mathcal{A}'_i$  with  $\tau = (s, \mathcal{H}, \kappa)$  is final if  $q \in F_i$  and  $i \in \mathcal{H}$ , that is, if it corresponds to a final local state of  $\mathcal{A}_i$  and must not take part in further phases. It is easy to check that each local final state of  $\mathcal{A}'_i$  is dead.

We fix some refined channel-state  $\rho'_0$  such that  $\pi'(\rho'_0) = \chi_0$ . A global state  $s' = (q', \rho')$  of  $S'$  is initial if  $\rho' = \rho'_0$  and there exists some initial global state  $s_0 \in I$  of  $S$  and some *secure* subset of instances  $\mathcal{H}_0$  such that for all  $i \in \mathcal{I}$  we have  $s' \downarrow i = (s_0 \downarrow i, \tau_0, \chi_0)$  where  $\tau_0 = (s_0, \mathcal{H}_0, \bar{0})$ .

With no surprise  $S'$  simulates any iteration of  $\mathcal{L}$ . To prove this basic fact we need to introduce some notations that relate the global states of  $S'$  to those of  $S$  in a natural way. First for any local state  $r = (q, \tau, \chi)$  we put  $\omega(r) = q$ . Second the first projection from  $A' = A \times T$  to  $A$  induces a mapping  $\omega$  from words over  $A'$  to words over  $A$ . Then any refined channel-state  $\rho'$  over  $A'$  corresponds to some refined channel-state  $\omega(\rho')$  such that  $\omega(\rho')(i, j) = \omega(\rho'(i, j))$  for all channels  $(i, j) \in \mathcal{K}$ . Finally each global state  $s' = (q', \rho')$  of  $S'$  corresponds to the global state  $\omega(s') = (q, \omega(\rho'))$  of  $S$  where  $q$  consists of the local states  $\omega(s' \downarrow i)$ .

**PROPOSITION 3.2.** *We have  $\mathcal{L}^+ \subseteq \mathcal{L}(S')$ .*

**Proof.** Let  $M_0 @ \chi_0, \dots, M_n @ \chi_0$  be some MSCs from  $\mathcal{L}$ . We show that the product  $M @ \chi_0 = M_0 @ \chi_0 \cdot \dots \cdot M_n @ \chi_0$  belongs to  $\mathcal{L}(S')$ . For each  $m \in [0, n]$  there are two global states  $s_m$  and  $s'_m$  of  $S$  and  $u_m \in L(S, s_m, s'_m)$  such that  $s_m \in I$ ,  $s'_m \in F$ , and  $\pi(u_m) \in \text{LE}(M_m)$ . For each  $m \in [0, n]$  we denote by  $\mathcal{H}_m$  the set of instances that are not live in all  $s_{m+1}, \dots, s_n$ . In particular  $\mathcal{H}_n = \mathcal{I}$  and for all  $m \geq 1$  we have  $\text{Live}(s_m) \subseteq \mathcal{I} \setminus \mathcal{H}_{m-1}$ ,  $\mathcal{H}_{m-1} \subseteq \mathcal{H}_m$ , and  $\mathcal{H}_m \setminus \mathcal{H}_{m-1} \subseteq \text{Live}(s_m)$ . We let  $s'_0$  be the initial global state of  $S'$  that corresponds to  $s_0$  and  $\mathcal{H}_0$ . By induction over  $m \leq n$ , we can check that there exists a word  $u$  that corresponds to an execution of  $S'$  consisting of  $m$  phases associated with the secure choices  $(s_1, \mathcal{H}_1), \dots, (s_m, \mathcal{H}_m)$  and such that  $\pi'(u) = \pi(u_0) \dots \pi(u_m)$ . Moreover  $u$  leads  $S'$  from  $s'_0$  to some global state  $s'$  such that  $\omega(s')$  is a final global state of  $S$ . In the case  $m = n$ , we get that  $s' \downarrow i = (q_i, \tau_m, \chi_m)$  with  $q_i \in F_i$  for all processes  $i \in \mathcal{I}$ . Recall that  $\mathcal{H}_n = \mathcal{I}$ . Let  $i \in \mathcal{I}$  such that  $i \notin \mathcal{H}_0$ . Let  $m$  be the least integer such that  $i \in \mathcal{H}_m$ . Then  $i \in \text{Live}(s_m)$ ,  $i$  takes part in  $u_m$ , and  $\tau_m = (s_m, \mathcal{H}_m, \kappa_m)$ . Thus  $s' \downarrow i \in F'_i$  for all  $i \in \mathcal{I}$  and  $M @ \chi_0 \in \mathcal{L}(S')$ . ■

### 3.3 A technical lemma

Let  $s'_0$  be an initial global state of  $S'$  associated with  $s_0 \in I$ ,  $\mathcal{H}_0 \subseteq \mathcal{I}$  and  $\tau_0 = (s_0, \mathcal{H}_0, \bar{0})$ . Let  $s'$  be some global state of  $S'$  and  $u, v$  be two words over  $\Sigma_{\mathcal{I}}^{A'}$ . We say that  $u$  and  $v$  are equivalent w.r.t.  $s'_0$  and  $s'$  if

- $u \in L(S', s'_0, s')$  if and only if  $v \in L(S', s'_0, s')$  and
- $\pi'(u) \in \text{LE}(M)$  if and only if  $\pi'(v) \in \text{LE}(M)$  for all context MSCs  $M @ \chi_0$ .

We come to our key technical result. The latter formalizes that each execution of  $S$  is equivalent to a series of phases that simulate possibly incomplete executions of  $S$ .

**LEMMA 3.3.** *Let  $s'$  be a global state of  $S'$  and  $u \in L(S', s'_0, s')$ . Let  $M @ \chi_0$  be the context MSC such that  $\pi'(u) \in \text{LE}(M)$ . There exist  $n \geq 0$ , a sequence of words  $u_0, \dots, u_n$  over  $\Sigma_{\mathcal{I}}^{A'}$ , a sequence  $s'_1, \dots, s'_{n+1}$  of global states of  $S'$  with  $s'_{n+1} = s'$ , a sequence of tags  $\tau_1, \dots, \tau_n$  with  $\tau_m = (s_m, \mathcal{H}_m, \kappa_m)$  for each  $m \in [1, n]$ , and a sequence  $\bar{s}_0, \dots, \bar{s}_n$  of global states of  $S$  such that*

- $(s_m, \mathcal{H}_m)$  is a secure choice for  $\mathcal{H}_{m-1}$  for all  $m \in [1, n]$ ,
- $u_0 \dots u_n$  is equivalent to  $u$  w.r.t.  $s'_0$  and  $s'$ ,

- $u_m \in L(\mathcal{S}', s'_m, s'_{m+1})$  and  $\pi'(u_m) \in \pi(L(\mathcal{S}, s_m, \bar{s}_m))$  for each  $m \in [0, n]$ ,
- if  $i$  takes part in  $u_m$  and  $m \in [0, n]$  then  $s'_{m+1} \downarrow i = (\bar{s}_m \downarrow i, \tau_m, \chi)$  for some  $\chi$ ,
- if  $i$  takes part in  $u_m$  and  $m \geq 1$  then  $\omega(s'_m) \downarrow i \in F_i$ ,
- if  $i$  takes part in  $u_l$  and  $i \in \text{Live}(s_m)$  with  $m \leq l$  then  $i$  takes part in  $u_m$ .

**Proof.** A phase  $m \in [0, n]$  is called incomplete if  $\bar{s}_m \notin F$ . We proceed by induction over the size of  $u$ . The base case where  $u$  is the empty word is trivial. Induction step: Assume  $u = v.a$  with  $a \in \Sigma_{\mathcal{I}}^A$ . The proof proceeds by case analysis over the five rules that define the transition relation of  $\mathcal{A}'_i$ . The key observation is the following. By induction hypothesis,  $\pi(v)$  is a linear extension of a prefix of some MSC from  $\mathcal{L}^+$ . Therefore there are at most  $B$  messages pending in  $s'$ . On the other hand there are at most  $|\mathcal{I}|$  instances  $i$  with  $\omega(s') \downarrow i \notin F_i$ . As a consequence incomplete phases in  $s'$  have distinct tags and no instance can skip any phase. ■

COROLLARY 3.4. *We have  $\mathcal{L}(\mathcal{S}') \subseteq \mathcal{L}^+$ .*

**Proof.** We apply Lemma 3.3 with the assumption that  $s'$  is a final global state. Then  $k \in \mathcal{H}_n$ ,  $\mathcal{H}_n = \mathcal{I}$  and  $\omega(s') \downarrow i \in F_i$  for all instances  $i$ . It follows that for all  $i \in \mathcal{I}$  and all  $m \in [0, n]$  we have  $\omega(s'_{m+1}) \downarrow i \in F_i$ . Furthermore  $i$  takes part in  $u_m$  whenever  $i \in \text{Live}(s_m)$ . Therefore  $\bar{s}_m$  is a *final* global state of  $\mathcal{S}$  hence  $\pi \circ \omega(u_m) \in \text{LE}(M_m)$  for some  $M_m @ \chi_0$  from  $\mathcal{L}$ . Since  $\pi'(u_m) = \pi \circ \omega(u_m)$ , we get that  $\pi'(u_0 \dots u_n) \in \text{LE}(M)$  with  $M @ \chi_0 \in \mathcal{L}^{n+1}$ . It follows that  $\pi'(u) \in \text{LE}(M)$ . Hence  $\mathcal{L}(\mathcal{S}') \subseteq \mathcal{L}^+$ . ■

COROLLARY 3.5. *The CFM  $\mathcal{S}'$  is safe.*

**Proof.** Let  $s'_0$  be an initial global state of  $\mathcal{S}'$  and  $s'$  be a global state of  $\mathcal{S}'$  reachable from  $s'_0$ . Let  $u \in L(\mathcal{S}', s'_0, s')$ . We apply Lemma 3.3 and consider  $u_0, \dots, u_n$  such that  $u_0 \dots u_n$  is equivalent to  $u$  w.r.t.  $s'_0$  and  $s'$ . The proof proceeds in two steps. We claim first that we can assume up to some completion of  $u$  that  $\omega(s'_{m+1}) \downarrow i = \bar{s}_m \downarrow i$  is a final local state of  $\mathcal{A}_i$  for all  $m \in [0, n]$  and all  $i \in \text{Live}(s_m)$ . This step makes use of the hypothesis that  $\mathcal{S}$  is safe: Each  $\omega(u_m)$  can be completed into a sequence that leads  $\mathcal{S}$  from  $s_m$  to a final global state. Second we proceed similarly to the proof of Proposition 3.2 and complete  $u$  in order to reach a final global state of  $\mathcal{S}'$ . This step makes use of the requirement that process  $k$  chooses always secure choices so that its local value of  $\mathcal{H}$  is secure after each phase. ■

## 4 Elementary decompositions of regular sets of MSCs

We come to the main result of this paper: *Any regular set of basic MSCs is accepted by a deadlock-free and stuck-free CFM with local termination.* The next statement expresses this result in the more general setting of context MSCs. Its proof follows from Lemma 2.1 and Theorem 3.1 by means of two simple inductions.

THEOREM 4.1. *All regular languages of context MSCs are implementable.*

**Proof.** Let  $\mathcal{L}$  be a regular set of context MSCs. By Lemma 2.1 we can assume that  $\mathcal{L}$  is located. We proceed by induction over the number of processes  $k$  that send messages

in  $\mathcal{L}$ . Base case: There are no send actions in any MSC from  $\mathcal{L}$ . Then  $\mathcal{L}$  is finite hence implementable. Induction step: We fix some instance  $k$  such that some MSCs from  $\mathcal{L}$  contain some send action from  $k$ . We consider the minimal deterministic automaton  $\mathcal{A} = (Q, \iota, F, \longrightarrow)$  over  $\Sigma_{\mathcal{I}}$  that accepts  $\text{LE}(\mathcal{L})$ . By Proposition 1.5 we can provide  $\mathcal{A}$  with a canonical mapping  $\chi$  which associates a channel-state  $\chi(q)$  to each state  $q \in Q$ . For any two states  $q, q' \in Q$  let  $\mathcal{L}_{q,q'}$  denote the set of context MSCs  $M @ \chi$  such that  $\chi = \chi(q)$  and  $q \xrightarrow{u} q'$  for all  $u \in \text{LE}(M)$ . Clearly  $\mathcal{L}_{q,q'}$  is a regular located set of MSCs. Moreover the subset of  $\mathcal{L}_{q,q'}$  that restricts to the context MSCs that are initiated by  $k$  (resp. do not contain any occurrence of any send action from  $k$ ) is also regular.

Now  $\mathcal{L}$  is a finite union of sets of context MSCs of the form  $\mathcal{L}_k = \mathcal{L}'_k \cdot \mathcal{L}''_k$  where all  $\mathcal{L}'_k$  and all  $\mathcal{L}''_k$  are located and regular, no send action from some  $k$  occurs in any  $\mathcal{L}'_k$ , and each  $\mathcal{L}''_k$  is initiated by  $k$  or consists of a single empty MSC. By induction hypothesis we can assume that  $\mathcal{L}$  is initiated by some  $k$ . For all  $q, q' \in Q$  and all  $j \in \mathcal{I}$ , the round  $\mathcal{L}_{q,q',j}$  is the subset of context MSCs  $M @ \chi$  from  $\mathcal{L}_{q,q'}$  that are initiated by  $k$  and contain a single send action from  $k$  and the latter is  $k!j$ . Clearly all rounds are regular. By induction hypothesis all rounds are implementable.

By Kleene's theorem,  $\mathcal{L}$  can be described by some rational expression  $r$  obtained from rounds by means of union, product, and strict iteration. Since  $\mathcal{L}$  is regular, it is channel-bounded and valid. It follows that any subexpression  $s$  of  $r$  describes a valid and channel-bounded set of MSCs. Moreover if  $s^+$  is a subexpression of  $r$  then  $s$  describes a located and initiated set of context MSCs whose domain and codomain coincide. By induction over the rational expression  $r$  with the help of Lemma 2.1 and Theorem 3.1, we get immediately that  $\mathcal{L}$  is implementable.  $\blacksquare$

## References

1. Alur R., Etesami K. and Yannakakis M.: *Realizability and verification of MSC graphs*. TCS **331** (2005) 97–114
2. Baudru N. and Morin R.: *Safe Implementability of Regular Message Sequence Charts Specifications*. Proc. of the ACIS 4th Int. Conf. SNBP (2003) 210–217
3. Bollig B. and Leucker M.: *Message-passing automata are expressively equivalent to EMSO logic*. TCS **358** (2006) 150–172
4. Bracho F., Droste M. and Kuske D.: *Representations of computations in concurrent automata by dependence orders*. TCS **174** (1997) 67–96
5. Diekert V. and Rozenberg G.: *The Book of Traces*. (World Scientific, 1995)
6. Droste M.: *Recognizable languages in concurrency monoids*. TCS **150** (1995) 77–109
7. Genest B., Kuske D., and Muscholl A.: *A Kleene theorem and model checking algorithms for existentially bounded communicating automata*. I&C **204** (2006) 920–956
8. Genest B., Muscholl A., Seidl H. and Zeitoun M.: *Infinite-State High-Level MSCs: Model-Checking and Realizability*. Journal of Computer and System Sciences **72** (2006) 617–647
9. Gunter E.L., Muscholl A. and Peled D.: *Compositional message sequence charts*. Intern. Journal on Software Tools for Technology Transfer **5(1)** (2003) 78–89
10. Henriksen J.G., Mukund M., Narayan Kumar K., Sohoni M. and Thiagarajan P.S.: *A Theory of Regular MSC Languages*. I&C **202** (2005) 1–38
11. Kuske D.: *Regular sets of infinite message sequence charts*. I&C **187** (2003) 80–109
12. Mukund M., Narayan Kumar K. and Sohoni M.: *Bounded time-stamping in message-passing systems*. TCS **290** (2003) 221–239
13. Zielonka W.: *Safe executions of recognizable trace languages by asynchronous automata*. LNCS **363** (1989) 278–289