Chapter 10

The Theory of Message Sequence Charts

K. Narayan Kumar
Chennai Mathematical Institute
H1 SIPCOT IT Park, Siruseri, India

Message Sequence Charts or MSCs are a visual formalism used in the specification of systems in many domains including telecommunications, object oriented design and forms a part of the UML language. Consequently, the formal study of MSCs has received considerable attention over the last decade. We survey some of the key results in this area with particular emphasis on the notion of regularity and its relationship to automata, logics and model-checking.

10.1. Message sequence charts

Message Sequence Charts (MSCs) [1] are an appealing visual formalism used in a number of software engineering notational frameworks such as SDL [2] and UML [3]. An MSC is a representation of a single behaviour or run of a system consisting of a collection of processes communicating with each other asynchronously via buffered channels. In this paper we shall restrict ourselves to systems with first-in-first-out (FIFO) channels. Fig. 10.1 shows an MSC involving three processes p, q and r and three messages. An MSC is to be read from top to bottom, the vertical lines denote processes and the arrows represent messages. Formally, MSCs are defined as labelled partial orders.

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite set of processes that communicate with each other through messages via reliable FIFO channels using a finite set of message types $\mathcal{M}$. For $p \in \mathcal{P}$, let $\Sigma_p = \{p!q(m), p?q(m) \mid p \neq q \in \mathcal{P}, m \in \mathcal{M}\}$ be the set of communication actions in which p participates. The action $p!q(m)$ is read as p *sends the message* m *to* q and the action $p?q(m)$ is read as p *receives the message* m *from* q. We set $\Sigma = \bigcup_{p \in \mathcal{P}} \Sigma_p$. We also denote the set of *channels* by $Ch = \{(p, q) \in \mathcal{P}^2 \mid p \neq q\}$. Whenever the set of processes $\mathcal{P}$ is clear from the context, we write Σ instead of $\Sigma_{\mathcal{P}}$, etc. Observe that our notation restricts us to a single channel from a process p to any other process q, however this is merely for notational convenience and the results do not change if we permit multiple channels between pairs of processes.

Labelled posets A $\Sigma_{\mathcal{P}}$-labelled poset is a structure $M = (E, \leq, \lambda)$ where $(E, \leq)$ is a partially ordered set and $\lambda : E \to \Sigma_{\mathcal{P}}$ is a labelling function. For $e \in E$, let

$\downarrow e = \{e' \mid e' \leq e\}$. For $p \in \mathcal{P}$ and $a \in \Sigma_{\mathcal{P}}$, we set $E_p = \{e \mid \lambda(e) \in \Sigma_p\}$ and $E_a = \{e \mid \lambda(e) = a\}$, respectively. For $(p, q) \in Ch$, we define the relation $<_{pq}$:

$$e <_{pq} e' \stackrel{\text{def}}{=} \exists m \in \mathcal{M} \text{ such that } \lambda(e) = p!q(m),\ \lambda(e') = q?p(m) \text{ and } |\downarrow e \cap E_{p!q(m)}| = |\downarrow e' \cap E_{q?p(m)}|$$

The relation $e <_{pq} e'$ says that channels are FIFO with respect to *each message*—if $e <_{pq} e'$, the message m read by q at e' is the one sent by p at e.

Finally, for each $p \in \mathcal{P}$, we define the relation $\leq_{pp} = (E_p \times E_p) \cap \leq$, with $<_{pp}$ standing for the largest irreflexive subset of $\leq_{pp}$. We write $Chn(e) = p!q$ if $\lambda(e) = p!q(m)$ and $Chn(e) = p?q$ if $\lambda(e) = p?q(m)$.

Definition 10.1. An MSC over $\mathcal{P}$ is a *finite* $\Sigma_{\mathcal{P}}$-labelled poset $M = (E, \leq, \lambda)$ where:

(1) Each relation $\leq_{pp}$ is a linear (total) order.
(2) If $p \neq q$ then for each $m \in \mathcal{M}$, $|E_{p!q(m)}| = |E_{q?p(m)}|$.
(3) If $e <_{pq} e'$, then $|\downarrow e \cap \left(\bigcup_{m \in \mathcal{M}} E_{p!q(m)}\right)| = |\downarrow e' \cap \left(\bigcup_{m \in \mathcal{M}} E_{q?p(m)}\right)|$.
(4) The partial order $\leq$ is the reflexive, transitive closure of $\bigcup_{p,q \in \mathcal{P}} <_{pq}$.

The second condition ensures that every message sent along a channel is received. The third condition says that every channel is FIFO across all messages.

In diagrams, the events of an MSC are presented in *visual order*. The events of each process are arranged in a vertical line and messages are displayed as horizontal or downward-sloping directed edges. Fig. 10.1 shows an example with three processes $\{p, q, r\}$ and six events $\{p_1, p_2, q_1, q_2, r_1, r_2\}$ corresponding to three messages—m_1 from p to q, m_2 from q to r and m_3 from p to r.

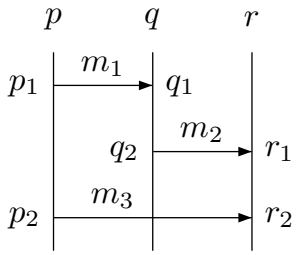

Fig. 10.1. An MSC

For an MSC $M = (E, \leq, \lambda)$, we let $\text{Lin}(M) = \{\lambda(\pi) \mid \pi \text{ is a linearization of } (E, \leq)\}$. For instance, $p!q(m_1)\ q?p(m_1)\ q!r(m_2)\ p!r(m_3)\ r?q(m_2)\ r?p(m_3)$ is one linearization of the MSC in Fig. 10.1. We write $\text{Lin}(M)$ for the set of linearizations of an MSC M.

Note that under the FIFO assumption an MSC can be reconstructed from any one linearization — the relation $<_{pp}$ is determined by the order of the p events in the linearization while $<_{pq}$ is determined by matching the i^{th} $p!q$ event with the i^{th} $q?p$ event.

Definition 10.2. Let M be an MSC and $B \in \mathbb{N}$. We say that $w \in \text{Lin}(M)$ is *B-bounded* if for every prefix v of w and for every channel $(p, q) \in Ch$, $\sum_{m \in \mathcal{M}} |v{\restriction}\{p!q(m)\}| - \sum_{m \in \mathcal{M}} |v{\restriction}\{q?p(m)\}| \leq B$, where $v{\restriction}\Gamma$ denotes the projection of v on $\Gamma \subseteq \Sigma_{\mathcal{P}}$.

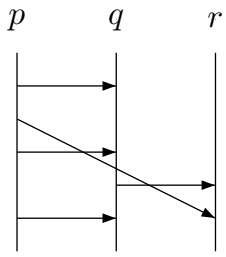

Fig. 10.2. A 3-bounded MSC

This means that along the sequential execution of M described by w, no channel ever contains more than B-messages. Consider the MSC in Fig. 10.2. The linearization $p!q\ q?p\ p!r\ p!q\ q?p\ q!r\ r?q\ p!q\ q?p\ r?p$ is 1-bounded while the linearization $p!q\ p!r\ p!q\ p!q\ q?p\ q?p\ p!r\ q?p\ r?q\ r?p$ is 3-bounded.

Definition 10.3. We say that M is *universally B-bounded* if every $w \in \mathrm{Lin}(M)$ is B-bounded and that M is *existentially B-bounded* if there is a $w \in \mathrm{Lin}(M)$ which is B-bounded. (We sometimes write B-bounded to mean universally B-bounded.)

The MSC in Fig. 10.2 is universally 3-bounded. The 1-bounded linearization listed earlier shows that this MSC is also existentially 1-bounded. The following proposition (paraphrased) from [4] characterizes universal B-boundedness for MSCs.

Proposition 10.1. *An MSC M is not B-bounded if and only if there are processes p, q and $p!q$ labelled events $e_1 <_{pp} e_2, \ldots <_{pp} e_{B+1}$ such that the $q?p$ labelled event e' with $e_1 <_{pq} e'$ is not below e_{B+1}.*

Optimal linearizations: Given an MSC M the smallest B for which it is B-bounded can be computed in linear time as shown in [5]. In particular, an optimal linearization w.r.t. boundedness can be computed by the following greedy strategy — at each step, extend the linearization, if possible, by a receive event; otherwise extend it by picking from the set of candidate send events the one that minimizes the maximum number of undelivered messages in any channel. Applying this greedy strategy to the MSC in Fig. 10.2 we get $p!q\ q?p\ p!r\ p!q\ q?p\ q!r\ r?q\ r?p\ p!q\ q?p$ which is a 1-bounded linearization.

10.1.1. *Concatenation of MSCs*

The concatenation of M_1 and M_2 written as $M_1 \circ M_2$ denotes the behaviour where each process participates in its events in M_1 and then follows it by participating in its events in M_2. It is not necessary for all processes to complete the events in M_1 before any process enters M_2.

Formally, the (asynchronous) concatenation of MSCs is defined as follows.

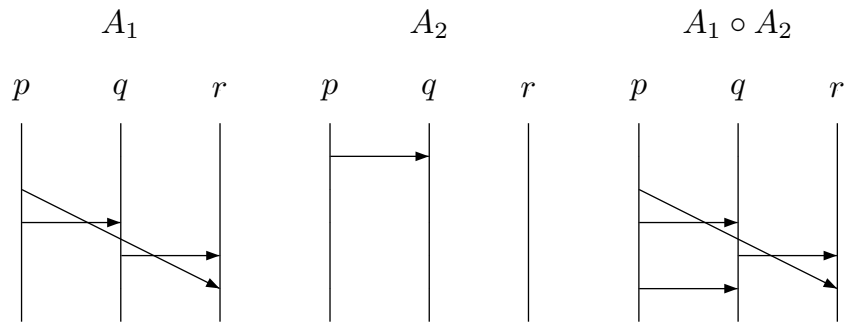

Fig. 10.3. MSC Concatenation

Definition 10.4. Let $M_1 = (E^1, \leq^1, \lambda_1)$ and $M_2 = (E^2, \leq^2, \lambda_2)$ be a pair of MSCs such that E^1 and E^2 are disjoint. The *(asynchronous) concatenation* of M_1 and M_2 yields the MSC $M_1 \circ M_2 = (E, \leq, \lambda)$ where $E = E^1 \cup E^2$, $\lambda(e) = \lambda_i(e)$ if $e \in E^i$, $i \in \{1, 2\}$, and $<_{pp} = <^1_{pp} \cup <^2_{pp} \cup (E^1_p \times E^2_p)$ and for $(p, q) \in Ch$, $<_{pq} = <^1_{pq} \cup <^2_{pq}$.

MSCs over a given alphabet form a monoid with the empty MSC as the identity (i.e. concatenation is an associative operation). Fig. 10.3 describes the concatenation of two MSCs. In the linearization $p!r\ p!q\ q?p\ q!r\ r?q\ p!q\ q?p\ r?p$ of $A_1 \circ A_2$ observe that the last event of A_1 occurs after both the events of A_2.

We can repeatedly decompose MSCs into the concatenation of smaller (nontrivial) MSCs till we are left with MSCs that cannot be decomposed.

Definition 10.5. An MSC M is said to be an *atom* if it cannot be expressed as the concatenation of two nontrivial MSCs.

The MSCs A_1 and A_2 in Fig. 10.3 are atoms. The MSC M in Fig. 10.2 is the concatenation $A_2 \circ A_1 \circ A_2$. In this case, the decomposition of M into atoms is unique. In general this need not be the case. To understand the exact nature of the decomposition of MSCs into atoms we need some terminology from the theory of traces.

10.1.2. *MSCs and Traces*

A *dependence alphabet* is a pair (Σ, D) where the alphabet Σ is a finite set of actions and the *dependence relation* $D \subseteq \Sigma \times \Sigma$ is reflexive and symmetric. The *independence relation* I is the complement of D. For $A \subseteq \Sigma$, the set of letters independent of A is denoted by $I(A) = \{b \in \Sigma \mid (a, b) \in I \text{ for all } a \in A\}$ and the set of letters depending on (some action in) A is denoted by $D(A) = \Sigma \setminus I(A)$.

A *Mazurkiewicz trace* is a labelled partial order $t = (V, \leq, \lambda)$ where V is a set of vertices labelled by $\lambda : V \to \Sigma$ and $\leq$ is a partial order over V satisfying the following conditions: For all $x \in V$, the downward set $\downarrow x = \{y \in V \mid y \leq x\}$ is finite, $(\lambda(x), \lambda(y)) \in D$ implies $x \leq y$ or $y \leq x$, and $x \lessdot y$ implies $(\lambda(x), \lambda(y)) \in D$, where $\lessdot = < \setminus <^2$ is the immediate successor relation in t.

Let $\equiv_I$ be the equivalence relation on Σ^* given by the reflexive transitive closure of the relation $uabv \sim ubav$ whenever $(a, b) \in I$. The set of linearizations of a trace t is an equivalence class of $\equiv_I$. Conversely, it is possible to reconstruct the trace from any given linearization. For a more detailed introduction to Trace theory, the reader is referred to **??**.

We are now in a position to characterize the different decompositions of an MSC into atoms. Let A be any finite collection of atoms over the set of processes $\mathcal{P}$. One can equip A with natural dependence alphabet structure (A, D) where $(a, b) \in D$ if and only if there is a process p that is active in both a and b.

Proposition 10.2. *Let M be an MSC, A be a finite set of atoms that contains the atoms that appear in M and (A, D) be the corresponding dependence alphabet. Then, if $M = a_1 \circ a_2 \circ \ldots a_k$ and $M = a'_1 \circ a'_2 \circ \ldots a'_l$ then $a_1 a_2 \ldots a_k \equiv_I a'_1 a'_2 \ldots a'_l$.*

Thus, the decomposition of an MSC into atoms is unique up to commutations of independent atoms. In particular, the set of atoms that are necessary to decompose an MSC M, denoted by $\mathcal{A}(M)$, is unambiguously defined. We shall write $\mathcal{A}_t(M)$ to denote the trace (or equivalently the linearizations of the trace) over atoms associated with M.

There is a second dependence alphabet associated with MSCs which yields an alternative characterization of B-bounded MSCs. Let $\Sigma_B = \Sigma_{\mathcal{P}} \times \{0, 1, \ldots, B-1\}$ for some natural number B. Let D be the dependence relation given by $(x, i)D(y, j)$ if either x and y occur in the same process or if $Chn(x) = p!q$, $Chn(y) = q?p$ and $i = j$. In a B-bounded MSC, the i^{th} receive on a channel must necessarily occur before the $(i+B)^{th}$ send on the same channel justifying the demand for an ordering between them in the dependency alphabet.

Given an MSC $M = (E, \leq, \lambda)$ we transform it into a Σ_B-labelled partial order $tr(M)$. Let $tr(M) = (E, \leq, \lambda')$ where $\lambda'(e) = (\lambda(e), i)$ where $i = |\{e' \mid e' < e,\ Chn(e') = Chn(e)\}|$. That is, we tag each event by its channel count modulo B. Does this yield a trace w.r.t the alphabet (Σ_B, D)?

There are two kinds of dependencies in the definition of the relation D. A dependency of the first kind, within a process, is also enforced in the MSC. An event labelled $p!q$ is guaranteed to be below the corresponding receive event labelled $q?p$ in the MSC. However, no ordering is necessary between other occurrences of events labelled by these two letters. Thus, the second kind of dependency demanded by D is in general stronger than the ordering in the MSC. Fig. 10.4 describes the trace $tr(M)$ over the

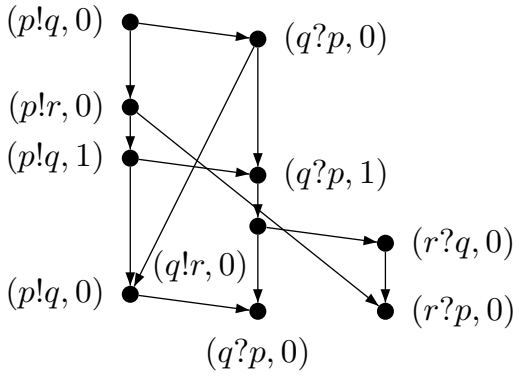

Fig. 10.4.

alphabet $\Sigma_{\mathcal{P}} \times \{0,1\}$ corresponding to the MSC in Fig. 10.2. Observe that there is no ordering between the first $q?p$ and the third $p!q$ in the MSC whilst they are ordered in the trace.

However, for B-bounded MSCS we have the following result.

Proposition 10.3. *[6; 7] Let M be an MSC. Then $tr(M)$ is a trace over the alphabet Σ_B if and only if M is a universally B-bounded MSC.*

Thus, universally B-bounded MSCs are traces over the aforementioned dependence alphabet. This allows us to exploit the well-developed theory of traces in the study of B-bounded MSCs.

Finally, let us examine existentially B-bounded MSCs and their relationship to traces. By definition, an MSC is not existentially B-bounded iff it cannot be linearized into a B-bounded word, and what rules out such a linearization is the following scenario — a $p!q$ event whose associated receive is above the next B $p!q$-labelled events under $\leq$. This can be formalized as follows.

Definition 10.6. Let $(E, \leq, \lambda)$ be a given MSC and B a natural number. The relation $<_{rev}$ on E, defined below, relates the i^{th} receive on a channel with the $(i+B)^{th}$ send on the same channel.

$$e <_{rev} e' \quad \stackrel{\text{def}}{=} \quad \exists f.\ f <_{pq} e \ \&\ \lambda(f) = \lambda(e') \ \&\ |\{e'' | f < e'' \leq e', \lambda(e'') = \lambda(f)\}| = B$$

Existentially B-bounded MSCs are precisely those that do not violate $<_{rev}$.

Proposition 10.4. *[8] Let $M = (E, \leq, \lambda)$ be an MSC and $<_{rev}$ be as defined above. M is existentially B-bounded if and only if $\leq \cup <_{rev}$ is acyclic. Existential B-boundedness of an MSC M can be decided in linear-time.*

In the MSC in Fig. 10.5, with $B = 2$, we find that the $<_{rev}$ edge marked with the dotted line induces a cycle. For $B = 3$, $<_{rev}$ is consistent with the ordering of this MSC implying that this MSC is existentially 3-bounded.

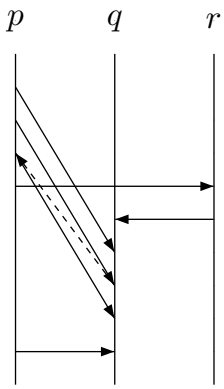

Fig. 10.5.

Let $M = (E, \leq, \lambda)$ and $\leq_B = (\leq \cup <_{rev})^*$. If M is existentially B-bounded then $(E, \leq_B, \lambda)$ is a labelled partial order. Let $tr'(M)$ be the labelled partial order $(E, \leq_B, \lambda')$ where λ' is the labelling function described earlier.

Proposition 10.5. *[8] If an MSC M is existentially B-bounded then the labelled partial order $tr'(M)$ is a trace over the alphabet (Σ_B, D). (If M is not existentially B-bounded then $tr'(M)$ is not even a partial order, leave alone a trace.)*

It worth noting that for a universally B-bounded MSC M, $<_{rev} \subseteq \leq$ and therefore $tr'(M) = tr(M)$. Thus, Prop. 10.5 can be thought of as a generalization of Prop. 10.3.

10.1.3. *MSC Languages and regularity*

An *MSC language* is a (finite or infinite) collection of MSCs over a given set of processes $\mathcal{P}$ and messages $\mathcal{M}$. Given the correspondence between MSCs and their linearizations, we may also regard a language of MSCs as a collection of words over $\Sigma_{\mathcal{P}}$ given by the linearizations of the MSCs in the language. In what follows we shall use these two notions interchangeably.

Definition 10.7. [9] A language of MSCs is said to be *regular* if the word language of its linearizations is a regular language.

In any prefix of a linearization of an MSC the number of $p!q$ events is at least as many as the number of $q?p$ events for any pair of processes p and q. We say that a word over $\Sigma_{\mathcal{P}}$ is *proper* if it satisfies this property. In any linearization of an MSC, there are as many $p!q$ events as there are $q?p$ events for every pair of processes p and q. We use *complete* to denote this property. Thus linearizations of MSCs are proper and complete words while their prefixes are proper words.

Let $(Q, \Sigma, \delta, s, F)$ be a deterministic finite automaton accepting a regular MSC language L. We further assume that every state is reachable and that a final state is reachable from every state. Suppose, u and v are two proper words that lead to the same state q from s. Let w be any word that leads from q to some final state. Thus, uw and vw are both complete words. Thus $|\#_{p!q}(v)| - |\#_{q?p}(v)| = |\#_{p!q}(v)| - |\#_{q?p}(v)|$ for each pair of processes p and q (where $\#_a(w)$ denotes the number of a's in the word w). This leads to the following result from [9] which assures us that regular languages are bounded.

Lemma 10.1. *Every regular MSC language L is B-bounded for some B. In particular, B can be chosen to be smaller than the size of the minimal automaton accepting the linearizations of L.*

The converse of this lemma is obviously false — for instance, consider the language $\{(p!q\ q?p)^i\ (r!s\ s?r)^i \mid i \geq 0\}$. We end this section with the definition of finitely generated MSC languages.

Definition 10.8. A language $\mathcal{L}$ of MSCs is said to be finitely generated if $\bigcup_{M \in \mathcal{L}} \mathcal{A}(M)$ is a finite set.

The MSC language given by the complete words $p!q\ (q!p\ p?q)^*\ q?p$ is not finitely generated. As a matter of fact, every word in this language is an atom.

10.2. Message Sequence Graphs

The ITU standard Z.120 describing MSCs also proposes a mechanism to describe collections of MSCs. This mechanism called HMSC (or High-level Message Sequence Charts) or Message Sequence Graphs (MSGs) allows branching, concatenation and iteration.

Definition 10.9. A *Message Sequence Graph* is a structure $\mathcal{G} = (Q, \rightarrow, Q_{in}, F, \Phi)$, where Q is a finite and nonempty set of states, $\rightarrow \subseteq Q \times Q$, $Q_{in} \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states and Φ labels each state with an MSC.

A *path* π through an MSG $\mathcal{G}$ is a sequence $q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ such that $(q_{i-1}, q_i) \in \rightarrow$ for $i \in \{1, 2, \ldots, n\}$. The MSC generated by π is $M(\pi) = M_0 \circ M_1 \circ M_2 \circ \cdots \circ M_n$, where $M_i = \Phi(q_i)$. A path $\pi = q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ is a *run* if $q_0 \in Q_{in}$ and $q_n \in F$. The language of MSCs accepted by $\mathcal{G}$ is $L(\mathcal{G}) = \{M(\pi) \mid \pi \text{ is a run through } \mathcal{G}\}$. We say that an MSC language $\mathcal{L}$ is *MSG-definable* if there exists an MSG $\mathcal{G}$ such that $\mathcal{L} = L(\mathcal{G})$.

An example of an MSG is depicted in Fig. 10.6. The initial state is marked $\Rightarrow$ and the final state has a double line. The MSC M corresponding to the path $q_0 \rightarrow q_1 \rightarrow q_0 \rightarrow q_2 \rightarrow q_0$ is also given in the figure.

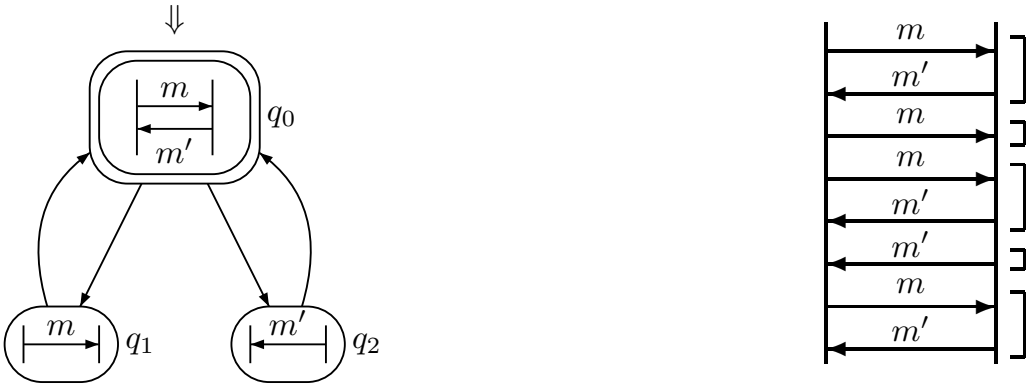

Fig. 10.6. A message sequence graph

It can be verified that the language generated by the MSG in Fig. 10.6 is a regular language. However, this need not always be the case. There are two reasons why MSGs can generate non-regular languages.

The first reason, as illustrated in Fig. 10.7, is its combination of concurrency and iteration. The language $\mathcal{L}$ generated by this MSG is $\{(M_1 \circ M_2)^n \mid n \geq 0\}$. The events in M_1 and M_2 are completely independent (concurrent) of each other. Thus, we may choose to linearize an MSC of the form $(M_1 \circ M_2)^n$ by first listing all the events involving p and q and then listing the events involving r and s. As a consequence, $\mathcal{L}$ projected to $\{p!q(m), r!s(m)\}^*$ consists of $\sigma \in \{p!q(m), r!s(m)\}^*$ such that $|\sigma\restriction_{p!q(m)}| = |\sigma\restriction_{r!s(m)}|$, which is not a regular string language. Hence $\mathcal{L}$ is not a regular MSC language.

The second reason, as illustrated by the producer-consumer example in Fig. 10.8, is that the buffers can be unbounded. The linearized language of this MSG is $\{w \mid \#_{p!q} w = \#_{q?p} w \;\&\; \forall v \leq w.\ \#_{p!q} v \geq \#_{q?p} v\}$.

However, the language of an MSG is always finitely generated since every MSC in the language can be decomposed using the MSCs that label the nodes of the MSG.

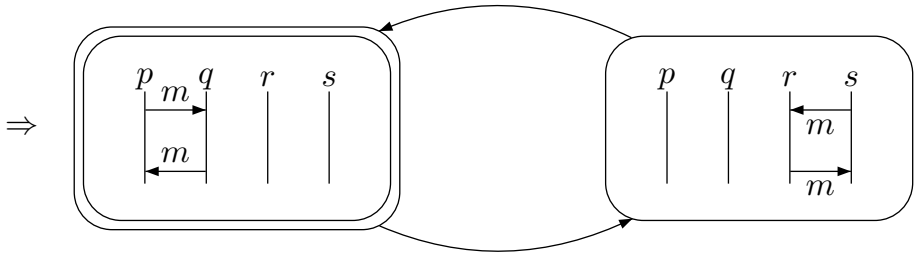

Fig. 10.7.

Proposition 10.6. *[10] Let $\mathcal{G}$ be a MSG. Let L_N be the set of MSCs that label the nodes of $\mathcal{G}$ and let $\mathcal{L}$ be the language generated by $\mathcal{G}$. Then, $\mathcal{L}$ is a finitely generated MSC language and in particular, $\mathcal{A}(\mathcal{L}) = \mathcal{A}(L_N)$.*

This also means that MSGs are not sufficient to describe every regular MSC language. For example, the language $p!r\ p!q\ q?p\ (q!r\ r?q\ r!q\ q?r)^*\ r?p$ is a regular MSC language that is not finitely generated and hence not MSG definable.

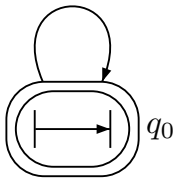

Fig. 10.8.

10.2.1. *Communication Graph*

The key to understanding the non-regularity of MSGs lies in studying their *communication graphs.*

Definition 10.10. For an MSC $M = (E, \leq, \lambda)$, CG_M, *the communication graph of* M, is the directed graph $(\mathcal{P}, \mapsto)$ where:

- $\mathcal{P}$ is the set of processes of the system.
- $(p, q) \in \mapsto$ iff there exists an $e \in E$ with $\lambda(e) = p!q(m)$.

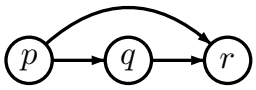

Fig. 10.9.

The communication graph of the MSC M in Fig. 10.2 is in Fig. 10.9. This graph is not strongly connected. This means that M^*, the iteration of M, is not a bounded language. The reasoning goes as follows: After participating in the events in the first copy of M, the process p can go ahead and participate in its events in the second copy and then the third copy and so on, before q or r participate in any event at all, forcing the channel from p to q to be unbounded. Suppose we modify the MSC M by adding a message from q to p. The iteration of M would still be unbounded — now the processes p and q can participate in all their events in copy one, and then copy two and so on before process r completes any event, forcing the channel from q to r to be unbounded. However the addition of an event from r to p would force the iteration of M to be bounded. But, this also makes the communication graph strongly connected.

Definition 10.11. A (communication) graph is *locally strongly connected* if the graph is the disjoint union of a collection of strongly connected components. An

MSG $\mathcal{G}$ is locally strongly connected if the communication graph of the MSC generated by every cycle (simple loop) in $\mathcal{G}$ is locally strongly connected.

The communication graph of every word in the language generated by the MSG in Fig. 10.7 is the same and is described in the Fig. 10.10. It is locally strongly connected and thus the MSG is locally strongly connected.

Suppose that the communication graph of an MSC M is locally strongly connected and let $X \subseteq \mathcal{P}$ be one of the strongly connected components. Let $p, q \in X$. Since there is a path from q to p in the communication graph of length at most $|X|$, in any MSC of the form $M^{|X|}$ (i.e. $M \circ M \circ \ldots \circ M$, $|X|$ times), there is p event which is above the q events in the first M. Thus, p can at most be in the $(|X|+1)^{st}$ copy before q completes its events in the first copy, ensuring that the channel from p to q is bounded. This argument does not rely on the fact that all the "copies" are identical but only uses the fact that all the copies have the same communication graph. Extending this argument gives

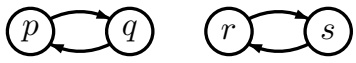

Fig. 10.10.

Lemma 10.2. *[11; 8] Let $\mathcal{G}$ be an MSG and M be an MSC.*

(1) If the communication graph of M is locally strongly connected then M^* is a B-bounded MSC language for $B \geq |M| \times |\mathcal{P}|$.
(2) If $\mathcal{G}$ is locally strongly connected then $L(\mathcal{G})$ is a B-bounded language for any $B \geq |\mathcal{G}|.|\mathcal{P}|.Max$ where Max is the maximum number of send events in a MSC labelling any one node of $\mathcal{G}$.
(3) If every node in $\mathcal{G}$ is reachable from an initial node and in turn can reach an accepting node and $L(\mathcal{G})$ is a bounded MSC language then $\mathcal{G}$ is locally strongly connected.

Boundedness, a necessary condition for regularity, by itself does not guarantee regularity — the MSG in Fig 10.7 has a locally strongly connected communication graph yet generates a non-regular language (In particular iterating $M_1 \circ M_2$ generates a non-regular language). A further structural restriction on MSGs is needed to rule out the other reason for non-regularity in MSG definable languages — iterations of concurrent behaviours.

Intuitively, the reason for the non-regularity of $(M_1 \circ M_2)^*$ is that $(M_1 \circ M_2)^N = M_1^N \circ M_2^N$, due to the independence of the events in M_1 and M_2, thus implicitly maintaining a counter.

Suppose the communication graph of an MSC M consists of a single nontrivial (i.e. of size at least 2) strongly connected component and a collection of trivial components (corresponding to each process that does not participate in any event in M). If p and q are two processes that participate in M, then in any segment of M^N of the form $M^{|\mathcal{P}|+1}$ there is a p event that depends on a previous q event within the segment and vice versa. Thus all independence is within small segments

(of size $M^{|\mathcal{P}|+1}$ or less) and the buffers are also bounded. This is sufficient ([12]) to ensure that M^* is a regular MSC language. This leads to the following definition.

Definition 10.12. An MSC M is said to be *locally synchronized* if its communication graph contains only one nontrivial strongly connected component and a collection of isolated vertices.

An MSG is said to be *locally synchronized* if the communication graph of the MSC generated by any loop in the MSG is locally synchronized. (Locally synchronized MSGs have also been called *bounded* MSGs or *com-connected* MSGs in literature.)

The following result shows that local synchronization is sufficient to guarantee regularity.

Lemma 10.3. *[13; 14] The language of any locally synchronized MSG is regular.*

Note that the definition of locally synchronized MSGs places a demand on all loops in the MSG and not just the cycles. Consider the MSG in Fig 10.11, adapted from [14]. The states q_0 and q_2 are labelled by the empty MSC. The states q_1, q_3 and q_4 are labelled by the MSCs M_1, M_3 and M_4 (described in the figure) respectively. Observe that every cycle in this MSG generates a locally synchronized MSC, however, the language accepted by this MSG is not regular. Every time the processes p and q switch from exchanging the message m to exchanging the message n (or vice versa) the processes r and s exchange a pair of messages. It is easy to derive the non-regularity from this observation.

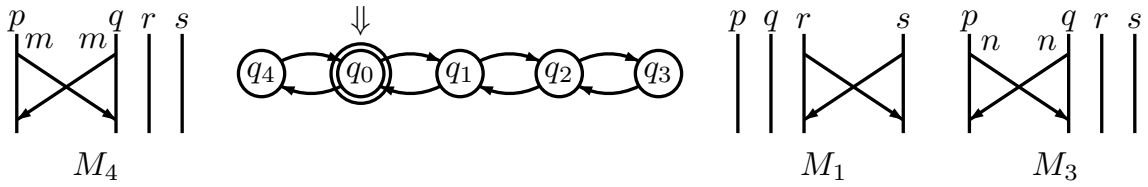

Fig. 10.11.

Interestingly, if we prohibit the labelling of nodes by the empty MSC, then the definition of a locally synchronized MSG can be weakened.

Proposition 10.7. *Let $\mathcal{G}$ be an MSG in which every node is labelled by a nontrivial MSC. If every cycle in $\mathcal{G}$ describes a locally synchronized MSC then $\mathcal{G}$ is locally synchronized.*

This can be seen as follows: Note that every loop that is not a cycle must properly contain a cycle. Let $p_0p_1 \dots p_k = q_0q_1 \dots q_m = p_{k+m} \dots p_0$ be a loop that contains the cycle $q_0q_1 \dots q_m$. The loop $p_0p_1 \dots p_kp_{k+m+1} \dots p_0$ is smaller and by the induction hypothesis generates a locally synchronized MSC. So does the cycle $q_0q_1 \dots q_m$. Any process that participates in the MSC labelling q_0 (and there is at

least one such process) is in the single nontrivial SCC of the communication graph of the smaller loop as well as the single nontrivial SCC of the communication graph of the cycle. Thus, the communication graph of the union of the loop and the cycle is also locally synchronized. Thus,

Lemma 10.4. *Let $\mathcal{G}$ be an MSG in which every node is labelled by a nontrivial MSC. If every cycle in $\mathcal{G}$ generates a locally synchronized MSC then the language of $\mathcal{G}$ is regular.*

Earlier, we remarked that not all regular MSC languages can be described using MSGs. The following result characterizes the collection of MSG definable regular MSC languages.

Lemma 10.5. *[10; 11] A regular MSC language L is definable using MSGs if and only if it is finitely generated. Any such language can also be described using a locally synchronized MSG.*

The proof in [11] (which incidentally defines MSGs as regular expressions constructed using MSCs) exploits the translation from MSCs to traces over the underlying set of atoms and pulls back the corresponding result for traces ([15]).

We have seen that locally strongly connected MSGs are guaranteed to be bounded but place no restrictions on independent iterations. Interestingly, we can also exclude the complications of independent iterations without forcing boundedness (or regularity).

Definition 10.13. [16; 17] An MSC is said to be *globally cooperative* if the symmetric closure of its communication graph is the union of a single strongly connected component and a collected of isolated vertices.

An MSG $\mathcal{G}$ is said to be *globally cooperative* if the MSC generated by any loop in $\mathcal{G}$ is globally cooperative. (In [17], globally cooperative MSGs are called *c-HMSCs*.)

The motivation for this definition comes from trace theory. A word w over a dependence alphabet (Σ, D) is said to be *connected* if the graph $(\Sigma_w, D \downarrow \Sigma_w)$, on the letters that appear in w and the dependency relation restricted to these letters, consists of a single connected component. Fix a dependence alphabet (Σ, D). A finite automaton over Σ in which every loop generates a connected word is said to be *loop connected*. Let the language L be the language of a loop connected finite automaton. Then, the language $\{tr(w) \mid w \in L\}$ of traces represented by words in L is a regular trace language. Equivalently the *trace-closure* of L, $\{w' \mid \exists w \in L. w' \sim w\}$, is a regular language ([18]).

We can easily transform an MSG into an equivalent one where every state is labelled by an atom — simply replace each node labelled by the MSC $A_1 \circ A_2 \ldots \circ A_k$, by a sequence of k nodes each labelled by an atom. (This transformation takes a globally cooperative MSG to a globally cooperative MSG.) Thus, MSGs can be

thought of as finite automata over the alphabet of atoms where the states are labelled by letters instead of transitions.

Lemma 10.6. *Let $\mathcal{G}$ be an MSG labelled by atoms and let $\mathcal{A}(\mathcal{G})$ be the set of atoms labelling the nodes of $\mathcal{G}$. Then, $\mathcal{G}$ is globally cooperative if and only if it is loop connected as a finite automaton over the dependency alphabet $(\mathcal{A}(\mathcal{G}), D)$ where aDb whenever there is a process that is active in both a and b.*

This leads to the following regularity theorem for a globally cooperative MSGs. This proves to be a very useful tool in resolving a number of decision problems for globally cooperative MSGs.

Theorem 10.1. *[19] Let $\mathcal{G}$ be a globally cooperative MSG. Then, the language $\{\mathcal{A}_t(M) \mid M \in L(\mathcal{G})\}$ is a regular language and a finite automaton accepting this language with size at most $2^{|\mathcal{G}|.|\mathcal{P}|}$ can be constructed.*

The following result from [19] shows that the if you take away independent iteration what is needed to ensure regularity is the boundedness of channels.

Proposition 10.8. *An MSG $\mathcal{G}$ is locally synchronized if and only if it is globally cooperative and accepts a bounded language.*

10.2.2. *Decision problems*

The language of an MSG is nonempty if and only if there is a path in the MSG from a start state to a final state. Thus, emptiness is decidable.

By Lemma 10.2, to decide whether a MSG accepts a universally bounded language it suffices to check if it is locally strongly connected. Every MSG accepts an existentially bounded language: in particular, if all the MSCs labelling the nodes of an MSG $\mathcal{G}$ are existentially B-bounded, then the language of $\mathcal{G}$ is also existentially B-bounded. The paper by Lohrey and Muscholl [8] establishes a comprehensive collection of the decidability results for a variety boundedness problems for MSGs and MSCs. Most importantly they demonstrate lower bounds for a variety of problems.

Theorem 10.2. *Let $\mathcal{G}$ be an MSG.*

(1) $\mathcal{G}$ is always existentially bounded.
(2) Given $\mathcal{G}$ and B we can check whether $\mathcal{G}$ is existentially B-bounded in linear time ([8]).
(3) Checking whether $\mathcal{G}$ is universally bounded is decidable ([11]). This problem is co-NP complete ([8]).
(4) Given $\mathcal{G}$ and B checking whether $\mathcal{G}$ is universally B-bounded is co-NP complete. This problem is co-NP complete even if B is fixed to the constant 1 ([8]).

There are many factors that contribute to the size of an MSC or an MSG — the number of processes in $\mathcal{P}$, the size of the message alphabet $\mathcal{M}$ and the number

of events. Theorem 10.2 holds as it is even if $\mathcal{M}$ is a singleton set ([8]). More often than not, the number of events is likely to be several orders of magnitude larger than the number of processes. Fortunately, if we fix the size of $\mathcal{P}$, all the above problems become efficiently solvable.

Theorem 10.3. *[8] Fix a set $\mathcal{P}$ of processes. The problem of checking whether a MSG over $\mathcal{P}$ is universally bounded (universally B-bounded for a given B) is in NL.*

The following lower-bound for checking structural properties of MSGs is from in [19]. Note that this result uses the assumption that the number of processes is part of the input.

Proposition 10.9. *Deciding whether a given MSG is locally synchronized (globally cooperative) is co-NP complete.*

Local synchronization provides a sufficient condition for regularity. However, there is no hope of obtaining an exact characterization.

Proposition 10.10. *[4; 11] Checking whether an MSG accepts a regular language is an undecidable problem.*

The reason for this undecidability is the following: It is easy to translate a dependency alphabet (Σ, D) into a collection of atoms $\mathcal{A}$ over a set of processes $\mathcal{P}$ by assigning an atom $\mathcal{A}(a)$ for each letter $a \in \Sigma$ in such a way that aDb if and only if $\mathcal{A}(a)$ and $\mathcal{A}(b)$ share an active process. For instance, the MSCs M_1, M_3 and M_4 (from Fig. 10.11) can be used to represent the dependence alphabet $(\{a, b, c\}, (b, c))$. Using this representation we can transform finite automata over Σ into MSGs over $(\mathcal{A}, D)$. The language of this MSG is regular if and only if the trace-closure of the original language is regular. However, checking the regularity of trace-closure is in general an undecidable problem (see for eg. [20; 21]).

Model-checking: A specification describes a collection of behaviours. These could be a set of allowed behaviours that the system should conform to, or a set of disallowed behaviours that the system must avoid. This results in two versions of the model checking problem, the positive and negative model checking problems. In the positive model checking problem the task is to verify that the set of behaviours of the system, L_{sy}, is a subset of the set of behaviours, L_{sp}, described by the specification. In the negative model checking problem, the task is to verify that $L_{sy} \cap L_{sp} = \emptyset$.

Suppose the specification as well as the system are described by MSGs.

Theorem 10.4. *[22; 13] For MSGs the following problems are undecidable:*

(1) Given $\mathcal{G}_1$ and $\mathcal{G}_2$, is $L(\mathcal{G}_1) \subseteq L(\mathcal{G}_2)$?

(2) Given $\mathcal{G}_1$ and $\mathcal{G}_2$, is $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$?

Once again, translations from the corresponding problems in trace theory suffices to prove the undecidability. As a matter of fact if we rule out independent iterations these problems become decidable.

Theorem 10.5. *[22] For globally cooperative MSGs the following problems are decidable:*

(1) Given $\mathcal{G}_1$ and $\mathcal{G}_2$, is $L(\mathcal{G}_1) \subseteq L(\mathcal{G}_2)$?
(2) Given $\mathcal{G}_1$ and $\mathcal{G}_2$, is $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$?

The proofs exploit the regular representation via atoms provided by Theorem 10.1 — $L(\mathcal{G}_1) \subseteq L(\mathcal{G}_2)$ if and only if $\mathcal{A}_t(L(\mathcal{G}_1)) \subseteq \mathcal{A}_t(L(\mathcal{G}_2))$ and $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$ if and only if $\mathcal{A}_t(L(\mathcal{G}_1)) \cap \mathcal{A}_t(L(\mathcal{G}_2)) = \emptyset$. Theorem 10.1 makes globally cooperative MSGs perhaps the most general of the classes of MSGs amenable to algorithmic analysis.

At this point we turn our attention to a regularity property that holds for all MSG definable languages.

Definition 10.14. A set X of linearizations is a *set of representatives* for an MSC language L if $\{M \mid \mathrm{Lin}(M) \cap X \neq \emptyset\} = L$.

Languages that have a regular set of representatives are needless to say interesting. If L is a regular MSC language then $\mathrm{Lin}(L)$ is a regular set of representatives for L. However, as we shall see, the class of languages with regular set of representatives is much larger.

Let $\mathcal{G}$ be an MSG. For each state $q \in \mathcal{G}$, fix a linearization w_q of the MSC labelling q. For any path $\pi = q_1 \to q_2 \to \ldots \to q_k$, let $w(\pi) = w_{q_1} w_{q_2} \ldots w_{q_k}$. Then, $\{w(\pi) \mid \pi$ is a run through $\mathcal{G}\}$ is a regular set as well as a set of representatives for $L(\mathcal{G})$.

Proposition 10.11. *[23] Every MSG definable language has a regular set of representatives.*

As such regular representations for L_{sy} and L_{sp} by themselves do not render the model-checking problems effective (as is clear from Theorem 10.4). Even if $L_1 \subseteq L_2$, it is easy to find representative sets X_1 and X_2 respectively in such a way that $X_1 \cap X_2 = \emptyset$. However,

Theorem 10.6. *[24] Suppose L_{sy} is given by a regular set of representatives X_{sy} and L_{sp} is a regular MSC language. Then, the positive and negative model checking problems are decidable. Thus, MSGs can be model-checked w.r.t. regular MSG specifications.*

In proof note that the positive model-checking problem boils down to checking if $X_{sy} \subseteq \mathrm{Lin}(L_{sp})$ which is merely the containment of regular languages, and the negative model-checking involves deciding if $X_{sy} \cap \mathrm{Lin}(L_{sp}) = \emptyset$.

An argument identical to the one used to prove Lemma 10.1 ensures that all the words in any regular representation of an MSC language are B-bounded for some B. Thus,

Proposition 10.12. *If L has a regular set of representatives then L is existentially B-bounded for some B.*

Let $\mathrm{Lin}^B(L)$ be the set of linearizations of L that are B-bounded. Theorem 10.6 can be strengthened as follows: from a regular set of representatives X_{sy} for the system we can derive a bound B such that every word in X_{sy} is B-bounded. Now, $X_{sy} \subseteq \mathrm{Lin}(L_{sp})$ if and only if $X_{sy} \subseteq \mathrm{Lin}^B(L_{sp})$ and $X_{sy} \cap \mathrm{Lin}(L_{sp}) = X_{sy} \cap \mathrm{Lin}^B(L_{sp})$. Thus, it suffices that $\mathrm{Lin}^B(L_{sp})$ be an (effectively constructible) regular set.

Theorem 10.7. *[24] Let L_{sy} be given by a regular set of representatives X_{sy} and let L_{sp} be such that $\mathrm{Lin}^B(L_{sp})$ is effectively regular for some B such that every word in X_{sy} is B-bounded. Then, the positive and negative model-checking problems are decidable.*

Later in this section we shall see that B-bounded linearizations of any globally cooperative MSG language is a regular language. Moreover, as we shall see in Section 10.4 there is another natural class of systems for which B-bounded linearizations are regular for any B. These results, drawn from [24; 25] and [19] show that the model-checking problem for MSGs is decidable for a fairly generous class of specifications.

10.2.3. *Compositional MSGs*

There have been several attempts at extending the definition of MSGs to increase their expressive power. For instance, the *netcharts* model ([26; 27]) attempts at combining the features of MSGs and petri-nets to obtain a model that can generate all regular MSC languages. In this section we consider a natural weakening of the definition of MSCs and MSGs that results in a richer specification language and yet retains most of the useful properties of MSGs.

A compositional MSC is essentially a segment of an MSC, and thus may contain receive events without matching send events and send events without matching receive events. Any association between sends and receives included must satisfy the FIFO assumption. Formally,

Definition 10.15. [28] A CMSC over $\mathcal{P}$ is a *finite* $\Sigma_{\mathcal{P}}$-labelled poset $M = (E, \leq, \lambda, msg)$ (with the notation defined in section 10.1) where

(1) Each relation $\leq_{pp}$ is a linear (total) order.

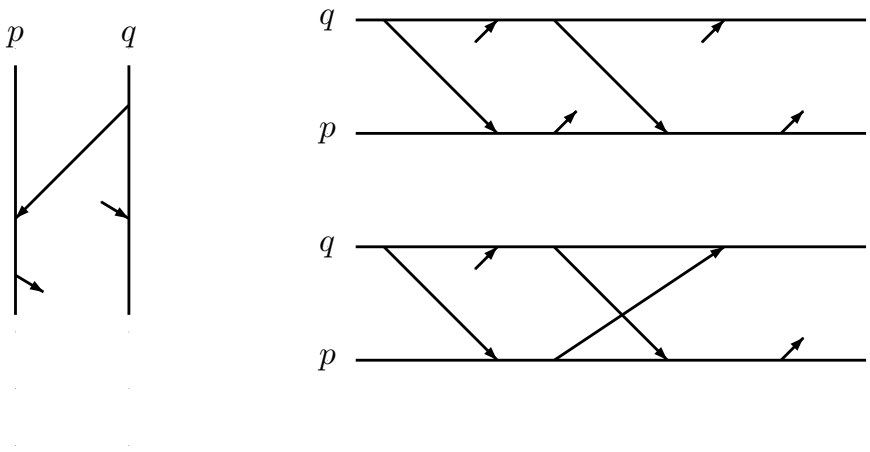

Fig. 10.12. A CMSC M and two elements of $M \circ M$

(2) msg is a partial injective mapping from S to R where

- $S = \{e \in E \mid \lambda(e) = p!q(m) \text{ for some } p, q, m\}$
- $R = \{e \in E \mid \lambda(e) = p?q(m) \text{ for some } p, q, m\}$

satisfying

(a) if $msg(s) = r$ then $s = p!q(m)$ and $r = q?p(m)$ for some p, q, m. We write $s <_{pq} msg(s)$ in that case.
(b) if $s_1 \leq_{pp} s_2$, $Ch(s_1) = Ch(s_2) = p!q$ and $msg(s_1)$ and $msg(s_2)$ are defined then $msg(s_1) \leq_{qq} msg(s_2)$.

(3) $\leq = (\cup_{p \in \mathcal{P}} \leq_{pp} \ \cup \ \cup_{(p,q) \in Ch} <_{pq})^*$.

In Fig. 10.12 we have a CMSC with one unmatched send and one unmatched receive events.

Every MSC is a CMSC. It is easy to check that a CMSC is an MSC if and only if msg is total and onto. Following [25], we define the concatenation of two CMSCs M_1 and M_2 as a set of CMSCs: for each process, the events in M_1 precede its events in M_2, further send events in M_1 may be matched with receive events in M_2, as long as it does not violate the FIFO condition. The result is a set as we are not obliged to match up unmatched sends in M_1 with unmatched receives in M_2 and there may be more than one way to match up unmatched sends in M_1 with unmatched receives in M_2.

Definition 10.16. Let $M_i = (E_i, \leq_i, \lambda_i, msg_i)$, $i = 1, 2$ be CMSCs with $E_1 \cap E_2 = \emptyset$. The concatenation $M_1 \circ M_2$ is the collection of CMSCs of the form $M = (E_1 \cup E_2, \leq, \lambda, msg)$ where

(1) $M \downarrow_{E_i} = M_i$ for $i = 1, 2$, where $M \downarrow_F$ is the restriction of $\leq$, λ and msg to the events in F.
(2) For each $e \in E_2$, if $e \leq e'$ then $e' \in E_2$ (i.e.) send events of E_2 cannot be matched with receive events in E_1.

Figure 10.12 illustrates CMSC concatenation by listing two CMSCs that belong to $M \circ M$. The FIFO assumption ensures that if $M_1 \circ M_2$ contains an MSC then it is unique. The operation $\circ$ can be extended to sets of CMSCs, $S \circ T = \{M \circ M' \mid M \in S \ \& \ M' \in T\}$. On sets of MSCs, the operation $\circ$ is associative, i.e., $S \circ (T \circ U) = (S \circ T) \circ U$. We generalize MSGs to CMSGs in the obvious manner.

Definition 10.17. [28] A *Compositional Message Sequence Graph* is a structure $\mathcal{G} = (Q, \to, Q_{in}, F, \Phi)$, where Q is a finite and nonempty set of states, $\to \subseteq Q \times Q$, $Q_{in} \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states and Φ labels each state with a CMSC.

A path $\pi = q_0 \to q_1 \to \cdots \to q_n$ is an *accepting run* if $q_0 \in Q_{in}$ and $q_n \in F$. The language of MSCs accepted by $\mathcal{G}$, $L(\mathcal{G})$, is the set of MSCs in the set

$$\{\Phi(q_0) \circ \Phi(q_1) \circ \ldots \circ \Phi(q_k) \mid q_0 \to q_1 \to \ldots q_k \text{ is an accepting run of } \mathcal{G}\}$$

We say that an CMSC language $\mathcal{L}$ is *CMSG-definable* if there exists an CMSG $\mathcal{G}$ such that $\mathcal{L} = L(\mathcal{G})$.

First of all note that some CMSGs may generate an empty MSC language even though there are paths from the initial state to final states. For instance, a CMSG with a single state that is initial and final, a self-loop, and labelled by the CMSC M from Fig. 10.12 is one such CMSG. Any CMSC generated by this CMSG has unmatched sends (and receives).

Observe that the CMSC $M_1 \circ M^i \circ M_2$, where M is the CMSC from Fig 10.12, M_1 is the CMSC with just a single $p!q$ event and M_2 is the CMSC with a single $q?p$ event, is an atom for all $i \geq 0$. Thus, CMSG recognizable languages need not be finitely generated. As a matter of fact, as shown in [28], it is quite easy to show that any regular MSC language is CMSG-definable — roughly speaking, we may replace each edge labelled $p!q$ (or $q?p$) in the finite automaton for this language by a node labelled by an MSC with a single $p!q$ (or $q?p$) event. This construction actually implies something stronger:

Proposition 10.13. *If L is an MSC language with a regular set of representatives then L is a CMSG-definable language.*

Interestingly, since every path in this CMSG corresponds to a valid linearization of an MSC in L, every path in this CMSG generates at least one MSC. Checking whether a CMSG generates any MSC or not is undecidable ([28]). Thus, CMSGs are somewhat unrestrained for a specification language.

Definition 10.18. [25] A CMSG $\mathcal{G}$ is said to be *safe* if for any accepting path $q_0 \to q_1 \to \ldots \to q_k$ the set $\Phi(q_0) \circ \Phi(q_1) \circ \ldots \circ \Phi(q_k)$ contains at least one MSC.

Every MSG is a safe CMSG. The property of being safe is decidable and safety gives a sufficient condition to ensure analyzability of CMSGs. Fix a linearization w_q for each node q of a safe CMSG $\mathcal{G}$. It is easy to see that the language

$\{w_{q_0} w_{q_1} \dots w_{q_k} \mid q_0 \to q_1 \to \dots q_k$ is an accepting run $\}$ is regular and a set of representatives of the language of $\mathcal{G}$. Combining this with the observation following Proposition 10.13 gives

Proposition 10.14. *[29] An MSC language L has a regular set of representatives if and only if it is the language of a safe CMSG. Thus, every safe CMSG language is existentially B-bounded for some B.*

Safe CMSGs should be as analyzable as the class of MSGs; after all, they enjoy the only regularity property that we have been able to associate with MSGs! To verify this, let us generalize some of the structural restrictions on MSGs to CMSGs.

Definition 10.19. [25] The *communication graph* of a CMSC is the directed graph whose vertices are the elements of $\mathcal{P}$ and there is an edge from p to q whenever there is a $p!q$ event and a $q?p$ event (the two need not necessarily be matched) in the CMSC.

The communication graph of the CMSC M in Fig. 10.12 is the complete directed graph on two vertices. The notions of *locally synchronized* CMSCs and *globally cooperative* CMSCs is defined as before. Finally, a CMSG $\mathcal{G}$ is *locally synchronized* if it is safe and every CMSC generated by every loop in $\mathcal{G}$ is locally synchronized. A CMSG $\mathcal{G}$ is *globally cooperative* if it is safe and every CMSC generated by every loop in $\mathcal{G}$ is globally cooperative.

Theorem 10.8. *[25] Let $\mathcal{G}$ be a globally cooperative CMSG and let B be an integer such that $B \geq |\mathcal{G}|$. Then, the set of B-bounded linearizations of $L(\mathcal{G})$ is a regular language (recognized by a finite automaton whose size is $O(|G|^{Poly(|G|,B,|\mathcal{P}|)})$.)*

The proof relies on the relationship to Mazurkiewicz traces. Combining this with Theorem 10.7 we get

Theorem 10.9. *[25] The positive and negative model checking problems are decidable when L_{sy} is the language of a safe CMSG and L_{sp} is the language of a globally cooperative CMSG.*

The essence of this story, which took some time to develop and culminate in the papers [19; 25], is the following: of the two suspected reasons for the non-analyzability of MSG (CMSG) based specifications, the culprit is independent (concurrent) iterations and eliminating that via a structural restriction (globally cooperative MSGs) delivers a generous decidability result for model-checking.

10.3. Monadic second order logic over MSCs

Monadic second order logic (or MSO) is the logical counter part to automata. Büchi and Elgot ([30; 31]) showed that MSO over finite words has the same expressive power as finite automata, Büchi then extended this to MSO over infinite words

and automata over infinite words that bear his name. These connections are not isolated and a host of similar connections have been established between MSO and automata. The most relevant of these results to our context is the one relating regular trace languages and MSO over traces due to Wolfgang Thomas ([32]) and extended to infinite traces by Ebinger and Muscholl ([33]).

Definition 10.20. Fix a set $\mathcal{P}$ of processes and messages. The formulas of the monadic second order logic over MSCs (MSO) are as follows:

$$\varphi \ ::= \ a(x) \mid x \in X \mid x \leq y \mid x \leq_{pp} y \mid x \lessdot_p y \mid x <_{pq} y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists X.\varphi \mid \exists x.\varphi$$

where $a \in \Sigma_{\mathcal{P}}$ and $(p, q) \in Ch$.

We will also be interested in the fragment of existential monadic second-order formulas (EMSO) which are of the form $\exists X_1 \exists X_2 \ldots \exists X_k.\ \varphi$ where φ is a first order formula. We will also be interested in restricted versions of these logics obtained by permitting only a subset of the 4 relational symbols in the syntax, and this will be made explicit by listing the allowed subset: for eg. $\mathrm{MSO}(\lessdot_p, <_{pq})$ to stand for the fragment that does not use $\leq$ and $\leq_{pp}$.

An MSO formula is interpreted over an MSC. The first order variables range over the events in the MSC, second order variables over sets of events and the relational operators have the obvious interpretation: $\leq$ is the ordering on the events of the MSC, $\leq_{pp}$ is the ordering on events in process p, $\lessdot_p$ is the immediate successor relation within a process p, and $<_{pq}$ is the message induced ordering between a send from p and the corresponding receive in q. Observe that the first two relations are ordering relations while the latter two are not. Finally $a(x)$ is true with $x = e$ if $\lambda(e) = a$. The interpretation of the logical operators and quantifiers is as usual. It is quite easy to see that $\leq_{pp}$ and $\lessdot_p$ can be defined using $\leq$, but it turns out that $<_{pq}$ cannot be so defined. Sentences in MSO define languages of MSCs, $L(\varphi) = \{M \mid M \models \varphi\}$ and in this case we say that L is MSO-definable.

Here is a sentence that characterises universally 2-bounded MSCs.

$$\bigwedge_{(p,q)\in Ch} \forall x.\forall y.\forall z.\ (p!q(x) \ \wedge \ p!q(y) \ \wedge \ p!q(z) \ \wedge \ (x < y) \ \wedge (y < z)) \implies \exists w.\ (x <_{pq} w) \ \wedge \ (w < z))$$

It asserts that in any sequence of 3 sends, the receive corresponding to the first send must be in the past of the third send (see Prop. 10.1). This can be generalized to describing universally B-bounded MSCs for any fixed B.

It is well-known that the transitive closure of a relation is definable in any monadic second order logic: xR^*y if and only if the smallest set containing x and closed under R also contains y. However, such a definition makes essential use of universal quantification over sets and consequently, such a translation is not always possible in the existential fragment of monadic second order logic. Since $\leq$ and

$\leq_{pp}$ can be defined as transitive closure of $\lessdot_p \cup <_{pq}$ and $\lessdot_p$ respectively, MSO over MSCs is equivalent to MSO($\lessdot_p, <_{pq}$).

In [9; 4] the following characterization theorem is presented.

Theorem 10.10. *A B-bounded language L is regular if and only if $L = L(\varphi) \cap \{M \mid M \text{is universally } B\text{-bounded}\}$ for some MSO($\leq$) formula φ.*

The result holds even if the logic is restricted to be EMSO($\leq$). The proof ([9; 4]) is based on the ideas used in similar results for MSO over traces in [32; 34]. In one direction, given a MSO($\leq$) formula φ, we construct a formula ψ in MSO over words such that $w \models \psi$ if and only if (i) w is a B-bounded complete and proper word and (ii) the MSC M_w generated by w satisfies φ and (iii) M_w is universally B-bounded. Establishing (i) and (ii) shows that the set of B-bounded linearizations of $L(\varphi)$ is a regular language (which implies Theorem 10.11 below).

Part (i) is easy as the set of B-bounded proper and complete words is a regular language and one may appeal to the Büchi-Elgot theorem. For part (ii), the proof proceeds by defining, in MSO over words, a binary relation $\preceq$ on the positions of the word in such a way that for any B-bounded w and positions i and j in w, $i \preceq j$ if and only if the corresponding events (say e_i and e_j) are ordered under $\leq$ in M_w. Clearly, $e_i \leq_{pp} e_j$ in M_w if and only if $i < j$ and i and j are p events in w. We still have to show that $<_{pq}$ is definable. This makes essential use of the B-boundedness of w. The formula asserts that there is a $k \in \{0, 1, \ldots, B-1\}$ such that i is labelled by $p!q$, the number of positions to its left labelled by $p!q$ is $k(\textit{modulo } B)$, j is to the right of i, it is a labelled by $q?p$ and the number of positions to its left labelled by $q?p$ is $k(\textit{modulo } B)$ and there is no position between i and j labelled by $q?p$ for which the number of positions to its left labelled $q?p$ is $k(\textit{modulo } B)$. Since transitive closure is definable in MSO the result follows. (The transitive closure can be avoided by a slightly more elaborate argument using the fact that one has to hop across processes at most $|\mathcal{P}|$ number of times.) Finally, part (iii) follows from the fact that with $\preceq$ and $\leq$, universally B-boundedness can be defined as explained in the example above.

The other direction is somewhat more involved. Given a regular MSC language and B, use Büchi-Elgot theorem to pick a formula ψ in MSO over words describing the linearizations of this language. Then show that in MSO($\leq$) one can define a relation $\preceq$ over the MSC that fixes a canonical linearization of the MSC (using techniques from [34]), and then interpret ψ over this linearization.

This result generalizes to MSO over infinite MSCs and regular languages of infinite MSCs as shown by D. Kuske ([7]). As a matter of fact, [7] provides a complete theory of regular languages of infinite MSCs. For the relationship between existentially B-bounded languages and MSO we have the following theorem:

Theorem 10.11. *[29] For any MSO formula φ and any B, $\text{Lin}^B(L(\varphi))$ is a regular language.*

An immediate consequence is that any existentially B-bounded language described by an MSO formula has a regular set of representatives. As a corollary to the previous two theorems we have

Corollary 10.1. *The problem of checking whether an MSO($\leq$) formula (or a MSO formula) is satisfiable over universally B-bounded MSCs is decidable. Similarly, checking satisfiability over existentially B-bounded MSCs is decidable.*

The natural question then is to ask "Can we model check safe CMSGs (or MSGs) w.r.t. to MSO?". The answer is affirmative. In proof note that, any safe CMSG has a regular set of representatives, Theorem 10.11 implies that $\mathrm{Lin}^B(L(\varphi))$ is regular for any B and any φ in MSO and thus Theorem 10.7 is applicable.

Theorem 10.12. *[23; 29] The problem of deciding whether every MSC generated by a safe CMSG (or MSG) satisfies a MSO formula is decidable.*

B.Bollig and M. Leucker [35] study the expressiveness of MSO and EMSO over MSCs and using techniques from [36] show that

Theorem 10.13.

(1) The monadic quantifier alternation hierarchy of the logic MSO (over MSCs) is infinite. Thus, MSO($\lessdot_p, <_{pq}$) is strictly more expressive than EMSO($\lessdot_p, <_{pq}$).
(2) The logics MSO($\leq$) and EMSO($\lessdot_p, <_{pq}$) are incomparable.

We shall return to the expressive power of MSO over MSCs a little later after we introduce an implementation model for MSCs.

10.4. Message Passing Automata

Safe CMSGs, MSGs and MSO are elegant and expressive languages to describe collections of MSCs. However, they are far removed from an execution model where each process is situated at a different location and there are limitations on what each process actually knows of the global state. The natural execution model for MSCs is that of *message passing automata* (also referred to as *communicating finite-state machines*).

A message passing automaton consists of a collection of finite state processes which communicate with each other by sending messages on FIFO channels. Each transition in a process involves either sending a message to some process or consuming a message from one of its input channels. It is possible to enrich these automata with local moves, however since it has no effect on the results in this section, we work without them. Fig. 10.13 illustrates a message passing automaton implementing a producer-consumer system and an MSC accepted by it. Formally, an MPA is defined as follows:

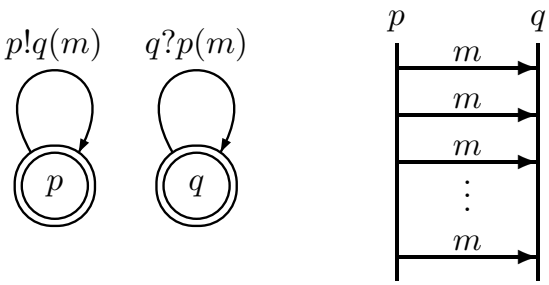

Fig. 10.13. An MPA

Definition 10.21. [37] Let $\Sigma_{\mathcal{P}}$ be the communication alphabet over the set of processes $\mathcal{P}$ and message alphabet $\mathcal{M}$. A *message-passing automaton (MPA)* over $\Sigma_{\mathcal{P}}$ is a structure $\mathcal{A} = (\{\mathcal{A}_p\}_{p\in\mathcal{P}}, \Delta, s_{in}, F)$ where:

- Δ is a finite alphabet of *auxiliary messages*.
- Each component $\mathcal{A}_p$ is of the form $(S_p, \rightarrow_p)$ where S_p is a finite set of p-local states and $\rightarrow_p \subseteq S_p \times \Sigma_p \times \Delta \times S_p$ is the p-local transition relation.
- $s_{in} \in \prod_{p\in\mathcal{P}} S_p$ is the global initial state.
- $F \subseteq \prod_{p\in\mathcal{P}} S_p$ is the set of global final states.

Observe that our definition allows the *tagging* of each message with auxiliary contents drawn from the set Δ.

The local transition relation $\rightarrow_p$ specifies how the process p sends and receives messages. The transition $(s, p!q(m), x, s')$ says that in state s, p can send the message m to q tagged with auxiliary information x and move to state s'. Similarly, the transition $(s, p?q(m), x, s')$ signifies that at state s, p can receive the message m from q tagged with information x and move to state s'.

A global state of $\mathcal{A}$ is an element of $\prod_{p\in\mathcal{P}} S_p$. For a global state s, s_p denotes the pth component of s. A *configuration* is a pair (s, χ) where s is a global state and $\chi : Ch \rightarrow (\mathcal{M} \times \Delta)^*$ is the *channel state* describing the message queue in each channel c. The *initial configuration* of $\mathcal{A}$ is $(s_{in}, \chi_\varepsilon)$ where $\chi_\varepsilon(c)$ is the empty string ε for every channel c. The set of *final configurations* of $\mathcal{A}$ is $F \times \{\chi_\varepsilon\}$. Observe that in a final configuration all the channels must be empty.

A global move of the automaton involves one of the process depositing a message into a channel (sending a message) or consuming a message from the channel (receiving a message) according to its local transition relation. Suppose, (s, χ) is a configuration and $(s_p, p!q(m), x, s'_p) \in \rightarrow_p$. Then, $(s, \chi) \stackrel{p!q(m)}{\Longrightarrow} (s', \chi')$ where for $r \neq p$, $s_r = s'_r$, for each $r \in \mathcal{P}$, $\chi'((p,q)) = \chi((p,q)) \cdot (m, x)$, and for $c \neq (p,q)$, $\chi'(c) = \chi(c)$. Similarly, if (s, χ) is a configuration and $(s_p, p?q(m), x, s'_p) \in \rightarrow_p$, then there is a global move $(s, \chi) \stackrel{p?q(m)}{\Longrightarrow} (s', \chi')$ where for $r \neq p$, $s_r = s'_r$, for each $r \in \mathcal{P}$, $\chi((q,p)) = (m, x) \cdot \chi'((q,p))$, and for $c \neq (q,p)$, $\chi'(c) = \chi(c)$.

A run of the automaton is a sequence of such global moves and we write $Conf_{\mathcal{A}}$

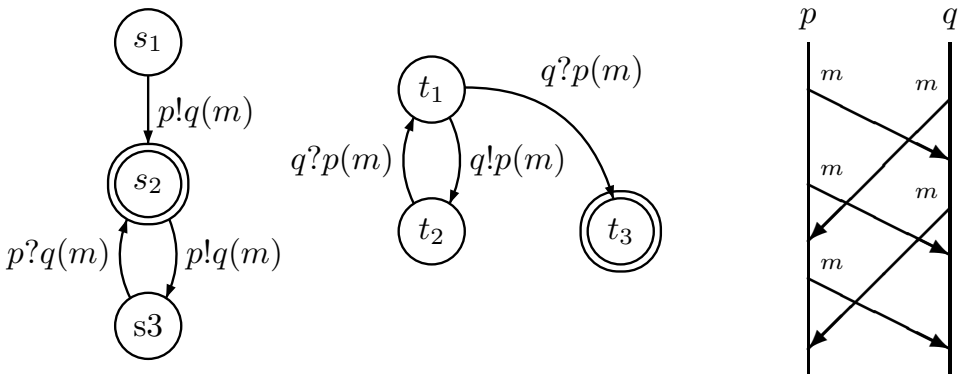

Fig. 10.14. An MPA accepting infinite number of atoms

for the set of reachable configurations of $\mathcal{A}$. A run is accepting if it ends in a final configuration. For instance

$$((p,q),\varepsilon) \stackrel{p!q(m)}{\Longrightarrow} ((p,q),m) \stackrel{p!q(m)}{\Longrightarrow} ((p,q),mm) \stackrel{q?p(m)}{\Longrightarrow} ((p,q),m) \stackrel{q?p(m)}{\Longrightarrow} ((p,q),\varepsilon)$$

is an accepting run the automaton in Fig. 10.13 on the word $p!q(m)p!q(m)q?p(m)q?p(m)$.

We define $L(\mathcal{A}) = \{\sigma \mid \mathcal{A} \text{ has an accepting run over } \sigma\}$. Since all channels are empty in the initial and final configurations and a message can be received only if it has already been sent, it is easy to check that any word accepted by an MPA is proper and complete. It is not difficult to see that if $L(\mathcal{A})$ contains one linearization of an MSC M then it contains all the linearizations of the MSC M. As a matter of fact, it is quite easy to define runs of MPAs directly on MSCs as a mapping from events on the MSC to global states of the automaton. Thus $L(\mathcal{A})$ is the set of linearizations of an MSC language. As usual we shall use $L(\mathcal{A})$ to denote the MSC language accepted by $\mathcal{A}$ as well as its linearizations.

Each configuration of an MPA records the messages sent and as yet undelivered. For $B \in \mathbb{N}$, a configuration (s,χ) is *B-bounded* if $|\chi(c)| \leq B$ for every channel $c \in Ch$ and $\mathcal{A}$ is a B-bounded automaton if every reachable configuration $(s,\chi) \in Conf_{\mathcal{A}}$ is B-bounded. Clearly a B-bounded automaton accepts a universally B-bounded language. The global state space of any B-bounded MPA is therefore finite and consequently, every B-bounded MPA accepts a regular MSC language. The converse is also true but we shall get to that a little later. The MPA in Fig. 10.13 accepts a unbounded language.

From any MPA $\mathcal{A}$ and a natural number B we can generate a finite automaton accepting precisely those B-bounded words that are accepted by $\mathcal{A}$ and thus

Proposition 10.15. *For any MPA $\mathcal{A}$ and any natural number B, the language* $\mathrm{Lin}^B(L(\mathcal{A}))$ *is a regular language.*

The language of an MPA need not be finitely generated. Fig. 10.14 describes an MPA and one of the MSCs it accepts. This MPA accepts a regular MSC language

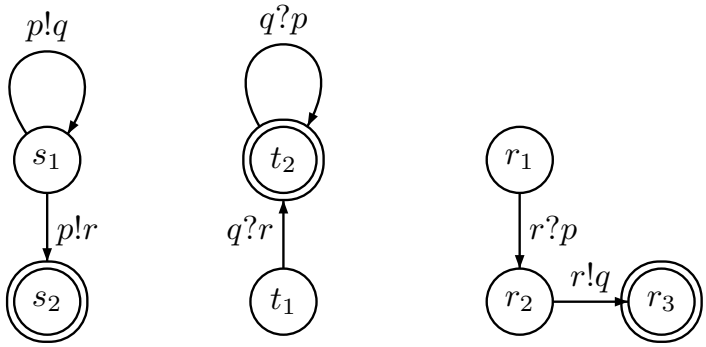

Fig. 10.15. An MPA with an existentially unbounded language

and every MSC accepted by this automaton is an atom. Thus, there are MPA acceptable languages that cannot be described by MSGs.

Finally, MPAs can also accept languages that are not existentially bounded (for any B), thus MPAs are capable of accepting languages that cannot be described using safe CMSGs. The MPA in Fig. 10.15 accepts the language of MSCs generated by the words $(p!q)^n\ p!r\ r?p\ r!q\ q?r\ (q?p)^n$ and it is easy to verify that this is not an existentially bounded language.

Definition 10.22. An MPA is said have local accepting states if $F = \prod_{p \in \mathcal{P}} F_p$ for some $F_p \subseteq S_p$.

10.4.1. *MPAs without auxiliary messages*

We first examine the power of MPAs without auxiliary message alphabets (i.e.) MPAs whose auxiliary alphabet is singleton.

Definition 10.23. An MSC language L is said to be *weakly realizable* if it is the language of an MPA with a singleton auxiliary message alphabet and with local accepting states.

Consider the language $\{M_1, M_2\}$ of MSCs (from Fig. 10.16). This set is not weakly realizable — Suppose $\mathcal{A}$ is an MPA accepting this language. From the accepting run on M_1, we know that there are runs $p_0 \xrightarrow{p!q(m)} p_1 \xrightarrow{p!s(m)} p_2$ and $q_0 \xrightarrow{q?p(m)} q_1$ for p and q ending in accepting states. Similarly, from the accepting run on M_2, we know that there are runs $r_0 \xrightarrow{r!s(m)} r_1$ and $s_0 \xrightarrow{s?r(m)} s_1 \xrightarrow{s?p(m)} s_2$ ending in accepting states. This means that the MSC M is also accepted, since p and q may behave exactly as they do in accepting M_1 and r and s may behave exactly as they do in accepting M_2 and all four processes end up in an accepting states on M.

The language of an MPA with local accepting states and over a singleton auxiliary message alphabet is merely the shuffle or free product of the local languages

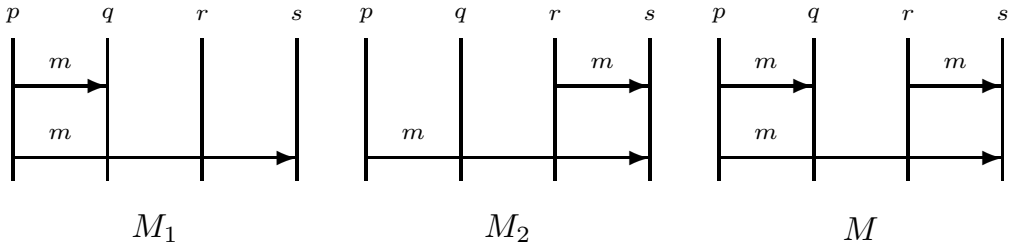

Fig. 10.16. An implied scenario

of the processes and this is formalized as follows:

Given an MSC M (or any linearization w of M) and a process p, $M\restriction p$ is the word over Σ_p consisting of the projection of M (or equivalently w) to the events in process p. For eg. $M_1\restriction p = p!q(m)p!s(m)$ for the MSC M_1 in Fig. 10.16. For a language L, $L_p = \{M\restriction p \mid M \in L\}$. Finally, given word languages L_p over Σ_p for each $p \in \mathcal{P}$, let $\prod_{p\in\mathcal{P}} L_p = \{M \mid M\restriction p \in L_p\}$ (the usual free product).

Definition 10.24. Given a set L of MSCs its *implied closure* $Imp(L)$ is defined as follows

$$Imp(L) \;=\; \{M \mid \forall p \in \mathcal{P}.\ \exists M_p \in L.\ M\restriction p = M_p\restriction p\}$$

If $M \in Imp(L) \setminus L$ then we say that L has *an implied scenario* and that M is an implied scenario of L.

In Fig 10.16, M is an implied scenario of $\{M_1, M_2\}$. The following characterization (albeit non-effective) is easy to prove.

Proposition 10.16. *[12] If L is weakly realizable then $L = Imp(L)$. Conversely, suppose L is an MSC language and $L\restriction p$ is a regular language for each p, then L is weakly realizable only if $L = Imp(L)$.*

Implied scenarios are of practical interest. Often, a designer specifies a system as a collection of MSCs using say an MSG. The existence of an implied scenario indicates that an implementation by MPAs (w/o auxiliary tagging) would result in behaviours not foreseen by the designer (these might or might not be *bad*). So it would be useful to check if a given MSC language L has any implied scenarios at all, and construct a representation for $Imp(L)$. The implied closure of a B-bounded language may contain MSCs that are not B-bounded. As

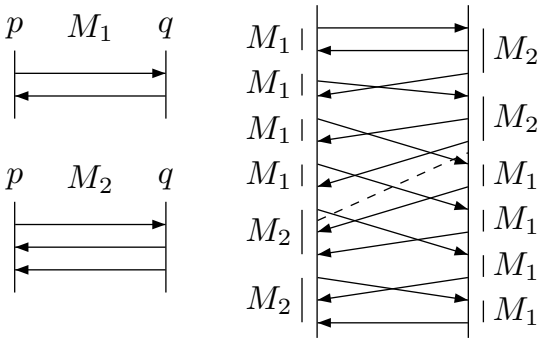

Fig. 10.17.

a matter of fact, the implied closure of a B-bounded regular language need not be bounded at all.

In Fig. 10.17 observe that the two MSCs M_1 and M_2 have complete communication graphs. Therefore, the language $(M_1 + M_2)^*$ is a regular MSG-definable language. On the other hand, for each $k \in \mathbb{N}$, the MSC in which the p-projection matches $M_1^{2k} M_2^k$ and the q-projection matches $M_2^k M_1^{2k}$ has a global cut where the channel (p, q) has capacity $k + 1$. The figure shows the case $k = 2$. The dotted line marks the global cut where the channel (p, q) has maximum capacity. Thus, $Imp(L)$ need not be a regular MSC language when L is a regular MSC language. It gets worse.

Theorem 10.14. *[38] The problem of checking whether $Imp(L) = L$ is undecidable even for regular MSC languages presented as locally synchronized MSGs.*

Let us examine this result a little. Let B be the bound on the channels in L. From the definition of $Imp(L)$ it is not difficult to check that if L is weakly realizable then an MPA $\mathcal{A}$ implementing L can be constructed as follows: For each process p pick a minimal finite automaton (w/o dead or unreachable states) accepting the language $L \restriction p$ as $\mathcal{A}_p$ and set $F = \prod_{p \in \mathcal{P}} F_p$. Thus we have a candidate implementation. Yet, weak realizability is undecidable because it is not possible to check whether the language accepted by $\mathcal{A}$ equals L. By restricting $\mathcal{A}$ to runs where no channel has more than B messages, we have a finite automaton accepting the set of B-bounded words in $Imp(L)$. Thus, checking if there is a B-bounded implied scenario for L is decidable. The difficulty is in finding if there are implied scenarios violating the B-bound. In particular, the existence of a partial run reaching a configuration where some channel has more than B messages does not mean that there are implied scenarios. This is because such a partial run may not be extendable to an accepting run. (However, this means that this partial run has ended in a configuration from where no final configuration is reachable. We shall return to this point a little later.)

Theorem 10.14 has been strengthened [39] to show that this problem is undecidable even when L is a 1-bounded language and undecidability holds even with just 2 processes. These undecidability arguments make essential use of the fact that the channels are FIFO (and therefore requires the message alphabet to have at least two messages.) Restriction to the trivial message alphabet yields the first positive result.

Proposition 10.17. *[17] The problem of checking whether the language of a locally synchronized MSG over a singleton message alphabet is weakly realizable is decidable.*

Since weak realizability is impossible to analyze, it is natural to look for stronger notions of implementability. Alur et al. propose a notion called *safe realizability* which is amenable to algorithmic analysis.

Definition 10.25. A configuration χ of an MPA $\mathcal{A}$ is a *deadlock* if there are no reachable final configurations. An MPA is said to be *deadlock-free* if it has no reachable deadlock configurations. A language is said to be *safely realizable* if it is the language of a deadlock-free MPA with local accepting states and over a singleton auxiliary message alphabet.

It is also possible to characterize safely realizable languages as a closure property akin to Prop. 10.16 (see [12; 40]).

Proposition 10.18. *L is safely realizable if and only if it is weakly realizable and satisfies the following closure property:*

$$\{w \mid \forall p.\exists u_p \in L.\ w\restriction p \leq u_p\} \subseteq \{w \mid \exists u \in L.w \leq u\}$$

The closure condition demands that any partial MSC (or proper word) whose projections on every process is consistent with some accepting run of the process must be extendable to an MSC (or proper complete word) in L. Fortunately, safe realizability is analyzable.

Theorem 10.15. *The problem of checking whether a given globally cooperative MSG generates a safely realizable language or not is decidable in EXPSPACE and the problem is EXPSPACE-complete even for locally synchronized MSGs. However, the safe realizability problem for arbitrary MSGs is undecidable.*

We go back to our analysis of why the availability of a candidate implementation does not suffice to ensure decidability of weak implementation. The analysis there ends with a conclusion placed within parenthesis. This conclusion stated in our recently acquired terminology states that the candidate implementation $\mathcal{A}$ has deadlocks or implied scenarios whenever it reaches a configuration violating the B-bound on some channel. Thus, if $\mathcal{A}$ ever reaches a configuration violating the B-bound on some channel, it cannot be a safe implementation of L. Finally, it is not difficult to verify, using Prop. 10.18 that if at all L is safely realizable then $\mathcal{A}$ is such a realization. That completes our sketch of the decidability argument.

The decidability for locally synchronized MSGs appears in [38], that for globally cooperative MSGs in [17] and the exact complexity result as well as the undecidability result is from [40].

10.4.2. *MPAs with auxiliary messages*

In this section we study the expressive power of MPAs with no restriction on the use of auxiliary message contents. We begin by showing that the collection $\{M_1, M_2\}$ from Fig. 10.16 can be implemented using auxiliary messages. An MPA with auxiliary messages drawn from the set $\{1, 2\}$ is described in figure 10.18 The process p signals the process s using the auxiliary information regarding the identity of the MSC (The auxiliary information in all the other messages can be ignored.)

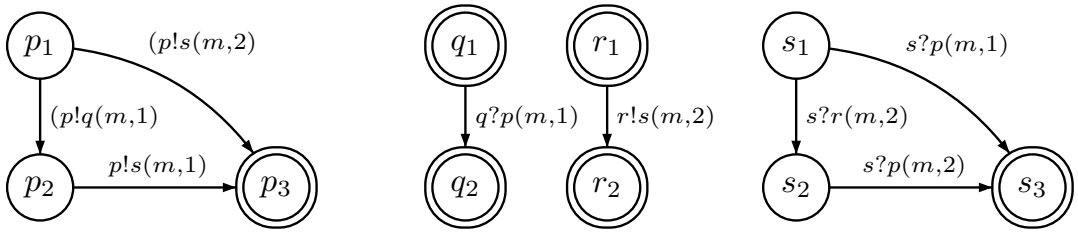

Fig. 10.18. An MPA accepting $\{M_1, M_2\}$

This example illustrates the use of auxiliary information: it allows a process to convey information about its past (in this case p conveys the information "I sent a message to q" to s by tagging the message with a 1 instead of a 2.) However, since the auxiliary message alphabet is a finite set, it only allows a bounded amount of information about the past to be conveyed in any message. It turns out that this ability to forward bounded amount of information is very powerful.

Theorem 10.16. *[9; 4] An MSC language L over a set of process $\mathcal{P}$ and messages $\mathcal{M}$ is regular if and only if there is an an MPA $\mathcal{A}$, over the same alphabet and with an auxiliary message alphabet Δ, such that $L = L(\mathcal{A})$.*

The proof of this theorem is beyond the scope of this paper. However we provide a brief sketch of the difficulties and main ideas involved.

This theorem is an example of a distributed synthesis theorem — it states that given the global description of a regular MSC language, it is possible to construct a distributed implementation as an MPA. The key ingredients that go into the proof are drawn from the celebrated result of W. Zielonka [41] showing that every regular trace language is recognized by an *asynchronous automaton*.

Let us examine the main difficulty in proving such a result. Suppose the global description is a finite automaton G. We can equip each local process with a copy of G if necessary. Yet, after a sequence of events w, which process is to keep track of the current state of G? Observe that every event takes place only in one process and each process directly observes only the events that it participates in, so it is not possible for any one process to maintain the state of G correctly. For the moment assume that we may tag the messages with unbounded amount of auxiliary information. Then, every process can send the entire history of all the events it has participated in as well as all the events about which it has learnt from others through messages it has received. So, whenever a message is received, the receiving process knows the entire set of events that are below this event in the MSC order. However, an MSC could have up to $|\mathcal{P}|$ maximal events and thus even

with this passing around of unbounded amount of information, no one process has information about the entire MSC.

Suppose the processes are $1, 2, \ldots K$. Process 1 has with it all the events that occur below the maximal 1 event in the MSC. Now, we would like process 2 to provide us with not all the events in its past, but only those that appear in its past but not in the past of process 1. For this, we need information about the events in each process $j \in \{2, 3, 4, \ldots, K\}$, that are in the past of 2 but not in the past of 1 (called the 1 residue at 2). With this information we can piece together all the events in the past of 1 and 2 without any ambiguity. Then, we need to obtain from process 3 information about events in $\{3, 4, \ldots, K\}$ that do not appear in the past of 1 and 2 (the $\{1, 2\}$ residue at 3) and so on. Then, the MSC can be reconstructed from the residues available at $1, 2, \ldots K$.

The reduction from unbounded to bounded information hinges on the fact that instead of keeping any partial MSC M (or a linearization w) of the history of a process, we might as well keep the transition function that this word or partial MSC defines on the state space of G.

The ability to program each process to maintain its residues w.r.t. to other processes requires a sophisticated time stamping algorithm from [42] which can perform comparisons such as "is my information about j more recent than i's information about j?". Most importantly this time-stamping algorithm requires each process to maintain only a bounded amount of information and tags each message with only a bounded amount of information. The proof in [43; 4] proceeds along these lines.

An alternative is to use the connection to regular trace languages, then use Zielonka's construction to obtain an asynchronous automaton and then translate back such an automaton into an MPA. This is the structure of the proof in [6]. The above theorem can be strengthened.

Definition 10.26. [43] An MPA is said to deterministic if

- If $(s, p!q, m_1, s_1') \in \longrightarrow_p$ and $(s, p!q, m_2, s_2') \in \longrightarrow_p$ then $m_1 = m_2$ and $s_1' = s_2'$.
- If $(s, p?q, m, s_1') \in \longrightarrow_p$ and $(s, p?q, m, s_2') \in \longrightarrow_p$ then $s_1' = s_2'$.

Determinacy requires that the nature of the message sent from p to q depends only on the local state of the sender p. Note, however, that from the same state, p may have the possibility of sending messages to more than one process. When receiving a message, the new state of the receiving process is fixed uniquely by its current local state and the content of the message. Once again, a process may be willing to receive messages from more than one process in a given state. This definition ensures that a deterministic automaton has at most one run on any $w \in \Sigma_{\mathcal{P}}^*$. Now, we are in a position to state the main characterization theorem for regular MSC languages.

Theorem 10.17. *[4] Let B be any integer and L be a language of universally B-bounded MSCs. Then the following are equivalent:*

(1) L is a regular MSC language
(2) L is definable in MSO($\leq$) (or EMSO($\leq$))
(3) L is the language of a deterministic B-bounded MPA.
(4) L is the language of a B-bounded MPA.

This theorem has been generalized to infinite MSCs by D.Kuske (see [7]). There are couple of other points worth noting: first of all the definition of MPAs uses a global set of final states and the physical realizability of such a global set of acceptance states is debatable. Secondly, the automata constructed in the proof of the above theorem may deadlock, and once again this makes its usability some what limited. In a recent series of papers, N. Baudru and R. Morin [44; 45], have shown that every regular MSC language can be implemented using (nondeterministic) MPAs that are deadlock free and whose acceptance set is local (i.e. $F = \prod_{p \in \mathcal{P}} F_p$ for some collection $(F_p)_{p \in \mathcal{P}}$.)

10.4.3. *Implementing existentially bounded languages*

The characterization in Theorem 10.17 shows that every regular MSC language can be implemented using a deterministic MPA. The corresponding question for MSC languages with a regular set of representatives was solved by Genest, Kuske and Muscholl [24; 25]. An earlier paper [19] set the stage for such a result through a host of results on the virtues of existentially bounded languages including the analyzability of globally cooperative CMSGs (Theorem 10.7) and efficient implementability (as MPAs) for a subclass (locally cooperative MSGs) of globally cooperative CMSGs. The proof of this characterization uses the translation to traces from existentially B-bounded MSCs (described in Section 10.1) to Mazurkiewicz traces.

Theorem 10.18. *[25] Let B be an integer and let L be a language of existentially B-bounded MSCs. Then the following statements are equivalent.*

(1) $\mathrm{Lin}^B(L)$ is a regular set of representatives for L.
(2) L is generated by a globally cooperative CMSG.
(3) L is MSO (EMSO) definable.
(4) L is implementable using an MPA.

We have already seen some of the relationships: Theorem. 10.8 shows that one can go from globally cooperative CMSGs to languages with $\mathrm{Lin}^B(L)$ as a regular set of representatives. Theorem 10.11 allows us to move from MSO definable languages to languages with $\mathrm{Lin}^B(L)$ as a regular set of representatives. Prop. 10.13 shows that any language with a regular set of representatives can be translated to a safe CMSG and Proposition 10.14 shows that every safe CMSG generates a language with a regular set of representatives. These relationships are strengthened to restrict the

class of CMSGs to globally cooperative CMSGs in [25] using Ochmanski's theorem ([15]). Finally by Prop. 10.15 the set $\mathrm{Lin}^B(L)$ is regular for any MPA.

This leaves the difficult part: a decomposition theorem showing that every existentially B-bounded language has a distributed implementation as an MPA and this involves a fairly complex argument via Mazurkiewicz trace theory and is beyond the scope of this article. The automaton constructed is a nondeterministic automaton and this is unavoidable.

Proposition 10.19. *[5] There are existentially bounded languages recognized by nondeterministic MPAs that cannot be recognized by deterministic MPAs.*

Translating MPAs to EMSO is quite routine, but the converse is not. A remarkable result due to Bollig and Leucker shows the following:

Theorem 10.19. *[35] An MSC language L is EMSO($\leqslant_p, <_{pq}$) definable if and only if it is the language of an MPA.*

In contrast to Theorems 10.17 and 10.18 this theorem applies to arbitrary MSCs. This theorem in combination with Theorem 10.13 shows that

Theorem 10.20. *[35] The class of languages accepted by MPAs is not closed under complementation.*

10.4.4. *Decision Problems*

Every channel is a queue and it is quite easy to simulate counter machines with MPAs. Consequently, general MPAs are too expressive for any sort of analysis:

Theorem 10.21. *[5]*

(1) The language emptiness problem for deterministic MPAs is undecidable.
(2) Given a B checking whether a deterministic MPA accepts a universally B-bounded language is undecidable for every $B > 0$.
(3) Given a deterministic, deadlock-free automaton in which every global state is accepting, checking whether it accepts a universally bounded language is undecidable.
(4) Given a B checking whether a deterministic MPA accepts a existentially B-bounded language is undecidable for every $B > 0$.
(5) Given a deterministic, deadlock-free automaton in which every global state is accepting, checking whether it accepts a existentially bounded language is undecidable.

Items (3) and (5) become decidable for a fixed B, since for a deadlock-free automaton it suffices to check if a configuration with $B+1$ messages in the channel is reachable.

10.5. Conclusion

In this article we have surveyed a selection of the results from the theory of MSCs. These are by no means exhaustive. A number of results that have been omitted here due to lack of space — to name a few, there is a host of decidability (and undecidability) results for the so called "'pattern matching" problems on MSCs (see for instance [46; 47]), the related problem of whether a class of MSCs can be implemented by MPAs with additional messages (not merely the addition of auxiliary content to existing messages; see for instance [48]), branching time specification and analysis (see for instance [49; 50; 51] and finally there have been some recent results in extending the theory of MSCs with time (see for instance [52; 53; 54]).

Acknowledgments

This work was partially supported by *Timed-DISCOVERI*, a project under the Indo-French Networking Project, the ARCUS Ile de France-Inde and the CMI-TCS Academic Alliance.

References

[1] ITU-TS. Itu-ts recommendation z.120: *Message Sequence Chart (MSC)*, (1997).
[2] E. Rudolph, P. Graubmann, and J. Grabowski, Tutorial on Message Sequence Charts, *Computer Networks and ISDN Systems.* **28**(12), 1629–1641, (1996).
[3] J. R. G. Booch, I. Jacobson, *Unified Modeling Language User Guide.* (Addison-Wesley, Reading, MA, 1997).
[4] J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. A. Sohoni, and P. S. Thiagarajan, A theory of regular MSC languages, *Information and Computation.* **202**(1), 1–38, (2005).
[5] B. Genest, D. Kuske, and A. Muscholl, On Communicating Automata with Bounded Channels, *Fundam. Inform.* **80**(1-3), 147–167, (2007).
[6] D. Kuske. A Further Step towards a Theory of Regular MSC Languages. In eds. H. Alt and A. Ferreira, *STACS*, vol. 2285, *Lecture Notes in Computer Science*, pp. 489–500. Springer, (2002). ISBN 3-540-43283-3.
[7] D. Kuske, Regular sets of infinite message sequence charts, *Information and Computation.* **187**(1), 80–109, (2003).
[8] M. Lohrey and A. Muscholl, Bounded MSC communication, *Information and Computation.* **189**(2), 160–181, (2004).
[9] J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. Regular Collections of Message Sequence Charts. In eds. M. Nielsen and B. Rovan, *MFCS*, vol. 1893, *Lecture Notes in Computer Science*, pp. 405–414. Springer, (2000). ISBN 3-540-67901-4.
[10] J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. On Message Sequence Graphs and Finitely Generated Regular MSC Languages. In eds. U. Montanari, J. D. P. Rolim, and E. Welzl, *ICALP*, vol. 1853, *Lecture Notes in Computer Science*, pp. 675–686. Springer, (2000). ISBN 3-540-67715-1.

[11] R. Morin. On Regular Message Sequence Chart Languages and Relationships to Mazurkiewicz Trace Theory. In eds. F. Honsell and M. Miculan, *FoSSaCS*, vol. 2030, *Lecture Notes in Computer Science*, pp. 332–346. Springer, (2001). ISBN 3-540-41864-4.

[12] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *ICSE*, pp. 304–313, (2000).

[13] R. Alur and M. Yannakakis. Model Checking of Message Sequence Charts. In eds. J. C. M. Baeten and S. Mauw, *CONCUR*, vol. 1664, *Lecture Notes in Computer Science*, pp. 114–129. Springer, (1999). ISBN 3-540-66425-4.

[14] A. Muscholl and D. Peled. Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces. In eds. M. Kutylowski, L. Pacholski, and T. Wierzbicki, *MFCS*, vol. 1672, *Lecture Notes in Computer Science*, pp. 81–91. Springer, (1999). ISBN 3-540-66408-4.

[15] E. Ochmanski, Regular behaviour of concurrent systems, *Bulletin of the EATCS*. **27**, 56–67, (1985).

[16] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-State High-Level MSCs: Model-Checking and Realizability. In eds. P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, *ICALP*, vol. 2380, *Lecture Notes in Computer Science*, pp. 657–668. Springer, (2002). ISBN 3-540-43864-5.

[17] R. Morin. Recognizable Sets of Message Sequence Charts. In eds. H. Alt and A. Ferreira, *STACS*, vol. 2285, *Lecture Notes in Computer Science*, pp. 523–534. Springer, (2002). ISBN 3-540-43283-3.

[18] M. Clerbout and M. Latteux, Semi-commutations, *Inf. Comput.* **73**(1), 59–74, (1987).

[19] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun, Infinite-state high-level MSCs: Model-checking and realizability, *J. Comput. Syst. Sci.* **72**(4), 617–647, (2006).

[20] A. Muscholl and H. Petersen, A Note on the Commutative Closure of Star-Free Languages, *Inf. Process. Lett.* **57**(2), 71–74, (1996).

[21] J. Sakarovitch. The "Last" Decision Problem for Rational Trace Languages. In ed. I. Simon, *LATIN*, vol. 583, *Lecture Notes in Computer Science*, pp. 460–473. Springer, (1992). ISBN 3-540-55284-7.

[22] A. Muscholl, D. Peled, and Z. Su. Deciding Properties for Message Sequence Charts. In ed. M. Nivat, *FoSSaCS*, vol. 1378, *Lecture Notes in Computer Science*, pp. 226–242. Springer, (1998). ISBN 3-540-64300-1.

[23] P. Madhusudan. Reasoning about Sequential and Branching Behaviours of Message Sequence Graphs. In eds. F. Orejas, P. G. Spirakis, and J. van Leeuwen, *ICALP*, vol. 2076, *Lecture Notes in Computer Science*, pp. 809–820. Springer, (2001). ISBN 3-540-42287-0.

[24] B. Genest, A. Muscholl, and D. Kuske. A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms. In eds. C. Calude, E. Calude, and M. J. Dinneen, *Developments in Language Theory*, vol. 3340, *Lecture Notes in Computer Science*, pp. 30–48. Springer, (2004). ISBN 3-540-24014-4.

[25] B. Genest, D. Kuske, and A. Muscholl, A Kleene theorem and model checking algorithms for existentially bounded communicating automata, *Inf. Comput.* **204**(6), 920–956, (2006).

[26] M. Mukund, K. Narayan Kumar, and P. S. Thiagarajan. Netcharts: Bridging the gap between HMSCs and executable specifications. In eds. R. M. Amadio and D. Lugiez, *CONCUR*, vol. 2761, *Lecture Notes in Computer Science*, pp. 293–307. Springer, (2003). ISBN 3-540-40753-7.

[27] N. Baudru and R. Morin. The Synthesis Problem of Netcharts. In eds. S. Donatelli

and P. S. Thiagarajan, *ICATPN*, vol. 4024, *Lecture Notes in Computer Science*, pp. 84–104. Springer, (2006). ISBN 3-540-34699-6.

[28] E. L. Gunter, A. Muscholl, and D. Peled. Compositional Message Sequence Charts. In eds. T. Margaria and W. Yi, *TACAS*, vol. 2031, *Lecture Notes in Computer Science*, pp. 496–511. Springer, (2001). ISBN 3-540-41865-2.

[29] P. Madhusudan and B. Meenakshi. Beyond Message Sequence Graphs. In eds. R. Hariharan, M. Mukund, and V. Vinay, *FSTTCS*, vol. 2245, *Lecture Notes in Computer Science*, pp. 256–267. Springer, (2001). ISBN 3-540-43002-4.

[30] J. R. Buechi, Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundl. Math.* **6**, 66–92, (1960).

[31] C. C. Elgot, Decision problems of finite automata design and related arithmetics., *Transactions of the American Mathematical Society.* **98**, 21–52, (1960).

[32] W. Thomas. Automata over infinite objects. In ed. J. van Leeuwen, *Handbook of Theoretical Computer Science, Volume B*, pp. 133–191. Elsevier, (1990).

[33] W. Ebinger and A. Muscholl, Logical Definability on Infinite Traces, *Theor. Comput. Sci.* **154**(1), 67–84, (1996).

[34] P. S. Thiagarajan and I. Walukiewicz. An Expressively Complete Linear Time Temporal Logic for Mazurkiewicz Traces. In *LICS*, pp. 183–194, (1997).

[35] B. Bollig and M. Leucker, Message-passing automata are expressively equivalent to EMSO logic, *Theor. Comput. Sci.* **358**(2-3), 150–172, (2006).

[36] O. Matz and W. Thomas. The Monadic Quantifier Alternation Hierarchy over Graphs is Infinite. In *LICS*, pp. 236–244, (1997).

[37] D. Brand and P. Zafiropulo, On Communicating Finite-State Machines, *J. ACM.* **30** (2), 323–342, (1983).

[38] R. Alur, K. Etessami, and M. Yannakakis. Realizability and Verification of MSC Graphs. In eds. F. Orejas, P. G. Spirakis, and J. van Leeuwen, *ICALP*, vol. 2076, *Lecture Notes in Computer Science*, pp. 797–808. Springer, (2001). ISBN 3-540-42287-0.

[39] P. Bhateja, P. Gastin, M. Mukund, and K. Narayan Kumar. Local Testing of Message Sequence Charts Is Difficult. In eds. E. Csuhaj-Varjú and Z. Ésik, *FCT*, vol. 4639, *Lecture Notes in Computer Science*, pp. 76–87. Springer, (2007). ISBN 978-3-540-74239-5.

[40] M. Lohrey, Realizability of high-level message sequence charts: closing the gaps, *Theor. Comput. Sci.* **309**(1-3), 529–554, (2003).

[41] W. Zielonka, Notes on Finite Asynchronous Automata, *ITA.* **21**(2), 99–135, (1987).

[42] M. Mukund, K. Narayan Kumar, and M. A. Sohoni, Bounded time-stamping in message-passing systems, *Theor. Comput. Sci.* **290**(1), 221–239, (2003).

[43] M. Mukund, K. Narayan Kumar, and M. A. Sohoni. Synthesizing Distributed Finite-State Systems from MSCs. In ed. C. Palamidessi, *CONCUR*, vol. 1877, *Lecture Notes in Computer Science*, pp. 521–535. Springer, (2000). ISBN 3-540-67897-2.

[44] N. Baudru and R. Morin. Safe Implementability of Regular Message Sequence Chart Specifications. In eds. W. Dosch and R. Y. Lee, *SNPD*, pp. 210–217. ACIS, (2003). ISBN 0-9700776-7-X.

[45] N. Baudru and R. Morin. Synthesis of Safe Message-Passing Systems. In eds. V. Arvind and S. Prasad, *FSTTCS*, vol. 4855, *Lecture Notes in Computer Science*, pp. 277–289. Springer, (2007). ISBN 978-3-540-77049-7.

[46] A. Muscholl. Matching Specifications for Message Sequence Charts. In ed. W. Thomas, *FoSSaCS*, vol. 1578, *Lecture Notes in Computer Science*, pp. 273–287. Springer, (1999). ISBN 3-540-65719-3.

[47] B. Genest and A. Muscholl. Pattern Matching and Membership for Hierarchical Mes-

sage Sequence Charts. In ed. S. Rajsbaum, *LATIN*, vol. 2286, *Lecture Notes in Computer Science*, pp. 326–340. Springer, (2002). ISBN 3-540-43400-3.

[48] B. Genest. On Implementation of Global Concurrent Systems with Local Asynchronous Controllers. In eds. M. Abadi and L. de Alfaro, *CONCUR*, vol. 3653, *Lecture Notes in Computer Science*, pp. 443–457. Springer, (2005). ISBN 3-540-28309-9.

[49] W. Damm and D. Harel, LSCs: Breathing Life into Message Sequence Charts, *Formal Methods in System Design*. **19**(1), 45–80, (2001).

[50] D. Harel and R. Marelly, Specifying and executing behavioral requirements: the play-in/play-out approach, *Software and System Modeling*. **2**(2), 82–107, (2003).

[51] D. Harel, H. Kugler, R. Marelly, and A. Pnueli. Smart Play-out of Behavioral Requirements. In eds. M. Aagaard and J. W. O'Leary, *FMCAD*, vol. 2517, *Lecture Notes in Computer Science*, pp. 378–398. Springer, (2002). ISBN 3-540-00116-6.

[52] S. Akshay, B. Bollig, and P. Gastin. Automata and Logics for Timed Message Sequence Charts. In eds. V. Arvind and S. Prasad, *FSTTCS*, vol. 4855, *Lecture Notes in Computer Science*, pp. 290–302. Springer, (2007). ISBN 978-3-540-77049-7.

[53] S. Akshay, M. Mukund, and K. Narayan Kumar. Checking Coverage for Infinite Collections of Timed Scenarios. In eds. L. Caires and V. T. Vasconcelos, *CONCUR*, vol. 4703, *Lecture Notes in Computer Science*, pp. 181–196. Springer, (2007). ISBN 978-3-540-74406-1.

[54] S. Akshay, B. Bollig, P. Gastin, M. Mukund, and K. Narayan Kumar. Distributed Timed Automata with Independently Evolving Clocks. In eds. F. van Breugel and M. Chechik, *CONCUR*, vol. 5201, *Lecture Notes in Computer Science*, pp. 82–97. Springer, (2008). ISBN 978-3-540-85360-2.