CONCUR 2000, 11th International Conference on Concurrency Theory
Proceedings: Catuscia Palamidessi (ed.)
Springer Lecture Notes in Computer Science **1877** (2000), 521–535.

Synthesizing distributed finite-state systems from MSCs*

Madhavan Mukund[1],
K. Narayan Kumar[1], and Milind Sohoni[2]

[1] Chennai Mathematical Institute, Chennai, India.
E-mail: {madhavan,kumar}@smi.ernet.in
[2] Indian Institute of Technology Bombay, Mumbai, India
E-mail: sohoni@cse.iitb.ernet.in

Abstract. Message sequence charts (MSCs) are an appealing visual formalism often used to capture system requirements in the early stages of design. An important question concerning MSCs is the following: how does one convert requirements represented by MSCs into state-based specifications? A first step in this direction was the definition in [9] of *regular* collections of MSCs, together with a characterization of this class in terms of finite-state distributed devices called message-passing automata. These automata are, in general, nondeterministic. In this paper, we strengthen this connection and describe how to directly associate a *deterministic* message-passing automaton with each regular collection of MSCs. Since real life distributed protocols are deterministic, our result is a more comprehensive solution to the synthesis problem for MSCs. Our result can be viewed as an extension of Zielonka's theorem for Mazurkiewicz trace languages [6, 19] to the setting of finite-state message-passing systems.

1 Introduction

Message sequence charts (MSCs) are an appealing visual formalism often used to capture system requirements in the early stages of design. They are particularly suited for describing scenarios for distributed telecommunication software [16]. They have also been called timing sequence diagrams, message flow diagrams and object interaction diagrams and are used in a number of software engineering methodologies [4, 7, 16]. In its basic form, an MSC depicts the exchange of messages between the processes of a distributed system along a single partially-ordered execution. A collection of MSCs is used to capture the scenarios that a designer might want the system to exhibit (or avoid).

Given the requirements in the form of a collection of MSCs, one can hope to do formal analysis and discover errors at an early stage. A standard way to

* This work has been supported in part by Project DRD/CSE/98-99/MS-4 between the Indian Institute of Technology Bombay and Ericsson (India), Project 2102-1 of the Indo-French Centre for Promotion of Advanced Research and NSF grant CDA9805735.

generate a collection of MSCs is to use a High Level Message Sequence Chart (HMSC) [12]. An HMSC is a finite directed graph in which each node is labelled, in turn, by an HMSC. The HMSCs labelling the vertices may not refer to each other. The collection of MSCs represented by an HMSC consists of all MSCs obtained by tracing a path in the HMSC from an initial vertex to a terminal vertex and concatenating the MSCs that are encountered along the path.

In order to analyze the collection of MSCs represented by an HMSC, one desirable property is that these MSCs correspond to the behaviour of a finite-state system. This property would be violated if the specification were to permit an unbounded number of messages to accumulate in a channel. A sufficient condition to rule out such *divergence* in an HMSC is described in [3]. Subsequently, it has been observed that HMSCs can also violate the finite-state property by exhibiting nonregular behaviour over causally independent bounded channels [2]. To remedy this, a stronger criterion is established in [2] which suffices to ensure that the behaviour described by an HMSC can be implemented by a (global) finite-state system. This leads to a more general question of when a collection of MSCs should be called regular. A robust notion of regularity has been proposed in [9]. As shown in [8], this notion strictly subsumes the finite-state collections generated by HMSCs. It turns out that the collections defined by HMSCs correspond to the class of finitely-generated regular collections of MSCs.

One of the main contributions of [9] is a characterization of regular collections of MSCs in terms of (distributed) finite-state devices called message-passing automata. This addresses the important synthesis problem for MSCs, first raised in [5]; namely, how to convert requirements as specified by MSCs into distributed, state-based specifications. The message-passing automata associated with regular collections of MSCs in [9] are, in general, nondeterministic. In this respect the solution to the synthesis problem in [9] is not completely satisfactory, since real life distributed protocols are normally deterministic.

In this paper, we strengthen the result of [9] by providing a technique for decomposing a sequential automaton accepting a regular collection of MSCs into a *deterministic* message-passing automaton. Our result can be viewed as the message-passing analogue of the celebrated theorem of Zielonka from Mazurkiewicz trace theory establishing that regular trace languages precisely correspond to the trace languages accepted by deterministic asynchronous automata [19].

In related work, a number of studies are available which are concerned with individual MSCs in terms of their semantics and properties [1, 10]. A variety of algorithms have been developed for HMSCs in the literature—for instance, pattern matching [11, 14, 15] and detection of process divergence and non-local choice [3]. A systematic account of the various model-checking problems associated with HMSCs and their complexities is given in [2].

The paper is organized as follows. In the next section we introduce MSCs and regular MSC languages. In Section 3 we define message-passing automata and state the problem. Section 4 describes a time-stamping result for message-passing systems from [13] which is crucial for proving our main result. In Section 5 we then introduce the notion of residues and show how the ability to locally compute residues would solve the decomposition problem. The next section describes a procedure for locally updating residues. This procedure is formalized as a message-passing automaton in Section 7.

2 Regular MSC Languages

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite set of processes which communicate with each other through messages. We assume that messages are never inserted, lost or modified—that is, the communication medium is reliable. However, there may be an arbitrary delay between the sending of a message and its receipt. We assume that messages are delivered in the order in which they are sent—in other words, the buffers between processes behave in a FIFO manner.

For each process $p \in \mathcal{P}$, we fix $\Sigma_p = \{p!q \mid p \neq q\} \cup \{p?q \mid p \neq q\}$ to be the set of communication actions in which p participates. The action $p!q$ is to be read as p *sends to* q and the action $p?q$ is to be read as p *receives from* q. We shall not be concerned with the actual messages that are sent and received—we are primarily interested in the pattern of communication between agents. We will also not deal with the internal actions of the agents. We set $\Sigma = \bigcup_{p \in \mathcal{P}} \Sigma_p$ and let a, b range over Σ.

For $a \in \Sigma$ and $u \in \Sigma^*$, $\#_a(u)$ denotes the number of occurrences of a in u. A word $u \in \Sigma^*$ is a *proper* communication sequence of the processes if for each prefix v of u and each pair of processes $p, q \in \mathcal{P}$, $\#_{p!q}(v) \geq \#_{q?p}(v)$—that is, at any point in the computation, at most as many messages have been received at q from p as have been sent from p to q. We say that u is a *complete* communication sequence if u is a proper communication sequence and for each pair of processes $p, q \in \mathcal{P}$, $\#_{p!q}(u) = \#_{q?p}(u)$—in other words, at the end of u, all messages that have been sent have also been received. We shall often say that u is proper (respectively, complete) to mean that u is a proper (respectively, complete) communication sequence over Σ.

Let $u = a_0 a_1 \ldots a_n \in \Sigma^*$ be proper. We can associate a natural Σ-labelled partial order $M_u = (E_u, \leq, \lambda)$ with u where:

- $E = \{(i, a_i) \mid i \in \{1, 2, \ldots, n\}$.
- $\lambda((i, a_i)) = a_i$. (If $\lambda(e) \in \Sigma_p$, we say that e is a p-event.)
- For $p, q \in \mathcal{P}$, we define relations $<_{pq} \subseteq E \times E$ as follows:
  - For $p \in \mathcal{P}$, $(i, a_i) <_{pp} (j, a_j)$ if $a_i, a_j \in \Sigma_p$, $i < j$ and there is no $i < k < j$ such that $a_k \in \Sigma_p$.
  - For $p, q \in \mathcal{P}$, $p \neq q$, $(i, a_i) <_{pq} (j, a_j)$ if $a_i = p!q$, $a_j = q?p$ and the sets $\{(k, a_k) \mid k < i, a_k = p!q\}$ and $\{(k, a_k) \mid k < j, a_k = q?p\}$ are of the same cardinality. Since messages are assumed to be read in FIFO fashion, $(i, a_i) <_{pq} (j, a_j)$ implies that the message read at the receive event (j, a_j) is the one sent at the send event (i, a_i).
- The partial order $\leq$ is the reflexive, transitive closure of the relations $\bigcup_{p,q \in \mathcal{P}} <_{pq}$.

We shall call the structure M_u generated from a *complete* communication sequence u a Message Sequence Chart (MSC).[1] The partial order between events in M_u is a more faithful representation of the causality between events in u than the sequential order induced by writing u as a string.

Henceforth, we shall implicitly associate with each proper word u the corresponding structure $M_u = (E_u, \leq, \lambda)$. In particular, E_u always refers to the set of events associated with the structure M_u generated from a proper word u. Abusing terminology, we refer to M_u as an MSC even if u is not complete.

Let $M_u = (E_u, \leq, \lambda)$ be an MSC. For $e \in E_u$, $e{\downarrow}$ denotes, as usual, the set $\{f \in E_u \mid f \leq e\}$. For $X \subseteq E_u$, $X{\downarrow}$ is defined to be $\bigcup_{e \in X} e{\downarrow}$.

We define a context-sensitive independence relation $I \subseteq \Sigma^* \times (\Sigma \times \Sigma)$ as follows: $(u, a, b) \in I$ provided that u is proper, $a \in \Sigma_p$ and $b \in \Sigma_q$ for distinct processes p and q, and, further, if $a = p!q$ and $b = q?p$ then $\#_a(u) > \#_b(u)$. Observe that if $(u, a, b) \in I$ then $(u, b, a) \in I$.

Let $\Sigma^\circ = \{u \in \Sigma^* \mid u \text{ is complete}\}$. We define $\sim \subseteq \Sigma^\circ \times \Sigma^\circ$ to be the least equivalence relation such that if $u = u_1abu_2$ and $u' = u_1bau_2$ and $(u_1, a, b) \in I$ then $u \sim u'$. It is important to note that $\sim$ is defined over Σ° (and not Σ^*).

The following simple observation shows that each MSC corresponds to a $\sim$-equivalence classes of complete communication sequences over Σ.

Proposition 2.1. *Let $u, v \in \Sigma^\circ$. Then, v is a linearization of M_u iff $u \sim v$.*

We define $L \subseteq \Sigma^*$ to be a *MSC language* if every member of L is complete and L is $\sim$-closed (that is, for each $u \in L$, if $u \in L$ and $u \sim v$ then $v \in L$.) We say that an MSC language L is a *regular* if L is a regular subset of Σ^*.

Given a regular subset $L \subseteq \Sigma^*$, we can decide whether L is a regular MSC language. We say that a state s in a finite-state automaton is *live* if there is a path from s to a final state. We then have the following result from [9].

Lemma 2.2. *Let $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$ be the minimal DFA representing L. Let $Chan = \{(p, q) \mid p, q \in \mathcal{P}, p \neq q\}$ denote the set of* channels. *L is a regular MSC language iff we can associate with each live state $s \in S$, a channel-capacity function $\mathcal{K}_s : Chan \to \mathbb{N}$ which satisfies the following conditions.*

(i) *If $s \in \{s_{in}\} \cup F$ then $\mathcal{K}_s(c) = 0$ for every $c \in Ch$.*
(ii) *If s, s' are live states and $\delta(s, p!q) = s'$ then $\mathcal{K}_{s'}((p, q)) = \mathcal{K}_s((p, q)) + 1$ and $\mathcal{K}_{s'}(c) = \mathcal{K}_s(c)$ for every $c \neq (p, q)$.*
(iii) *If s, s' are live states $\delta(s, q?p) = s'$ then $\mathcal{K}_s((p, q)) > 0$, $\mathcal{K}_{s'}((p, q)) = \mathcal{K}_s((p, q)) - 1$ and $\mathcal{K}_{s'}(c) = \mathcal{K}_s(c)$ for every $c \neq (p, q)$.*
(iv) *Suppose $\delta(s, a) = s_1$ and $\delta(s_1, b) = s_2$ with $a \in \Sigma_p$ and $b \in \Sigma_q$, $p \neq q$. If it is not the case that $a = p!q$ and $b = q?p$, or it is the case that $\mathcal{K}_s((p, q)) > 0$, there exists s'_1 such that $\delta(s, b) = s'_1$ and $\delta(s'_1, a) = s_2$.*

[1] Our definition captures the standard partial-order semantics associated with MSCs [1, 16]. See [9] for an equivalent definition of MSCs in terms of labelled partial orders.

Observe that the conditions described in the lemma can be checked in time linear in the size of δ.

Item (iv) of the lemma has useful consequences. As usual, we extend δ to words and let $\delta(s_{in}, u)$ denote the (unique) state reached by $\mathcal{A}$ on reading an input u. Let u be a proper word and let a, b be communication actions such that (u, a, b) belongs to the context-sensitive independence relation defined earlier. Item (iv) guarantees that $\delta(s_{in}, uab) = \delta(s_{in}, uba)$. From this, we can conclude that if v, w are complete words such that $v \sim w$, then $\delta(s_{in}, v) = \delta(s_{in}, w)$.

3 Message-passing automata

We now define distributed automata which accept MSC languages.

Message-passing automaton A *message-passing automaton* over Σ is a structure $\mathcal{A} = (\{\mathcal{A}_p\}_{p\in\mathcal{P}}, \mathcal{M}, s_{in}, \mathcal{F})$ where

- $\mathcal{M}$ is a finite alphabet of 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.
  - $\rightarrow_p \subseteq (S_p \times \Sigma_p \times \mathcal{M} \times S_p)$ is the p-local transition relation.
- $s_{in} \in \prod_{p\in\mathcal{P}} S_p$ is the global initial state.
- $\mathcal{F} \subseteq \prod_{p\in\mathcal{P}} S_p$ is the set of global final states.

The local transition relation $\rightarrow_p$ specifies how the process p sends and receives messages. The transition $(s, p!q, m, s')$ specifies that when p is in the state s, it can send the message m to q (by executing the communication action $p!q$) and go to the state s'. The message m is, as a result, appended to the queue of messages in the channel (p, q). Similarly, the transition $(s, p?q, m, s')$ signifies that at the state s, the process p can receive the message m from q by executing the action $p?q$ and go to the state s'. The message m is removed from the head of the queue of messages in the channel (q, p).

We say that $\mathcal{A}$ is *deterministic* if the local transition relation $\rightarrow_p$ for each component $\mathcal{A}_p$ satisfies the following conditions:

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

In other words, when a component $\mathcal{A}_p$ of a deterministic automaton $\mathcal{A}$ executes a send action, the current state of $\mathcal{A}_p$ uniquely determines the message sent as well as the new state of $\mathcal{A}_p$, and when $\mathcal{A}_p$ executes a receive action, the current state of $\mathcal{A}_p$ and the nature of the message at the head of the queue uniquely determine the new state of $\mathcal{A}_p$.

The set of global states of $\mathcal{A}$ is given by $\prod_{p\in\mathcal{P}} S_p$. For a global state s, we let s_p denote the pth component of s. A *configuration* is a pair (s, χ) where s is a global state and $\chi : \mathit{Chan} \rightarrow \mathcal{M}^*$ is the *channel state* which specifies the queue of messages currently residing 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 $\mathcal{F} \times \{\chi_\varepsilon\}$.

We now define the set of reachable configurations $Conf_\mathcal{A}$ and the global transition relation $\Rightarrow \subseteq Conf_\mathcal{A} \times \Sigma \times Conf_\mathcal{A}$ inductively as follows:

- $(s_{in}, \chi_\varepsilon) \in Conf_\mathcal{A}$.
- Suppose $(s, \chi) \in Conf_\mathcal{A}$, (s', χ') is a configuration and $(s_p, p!q, m, s'_p) \in \rightarrow_p$ such that the following conditions are satisfied:
  - $r \neq p$ implies $s_r = s'_r$ for each $r \in \mathcal{P}$.
  - $\chi'((p,q)) = \chi((p,q)) \cdot m$ and for $c \neq (p,q)$, $\chi'(c) = \chi(c)$.

  Then $(s, \chi) \stackrel{p!q}{\Longrightarrow} (s', \chi')$ and $(s', \chi') \in Conf_\mathcal{A}$.
- Suppose $(s, \chi) \in Conf_\mathcal{A}$, (s', χ') is a configuration and $(s_p, p?q, m, s'_p) \in \rightarrow_p$ such that the following conditions are satisfied:
  - $r \neq p$ implies $s_r = s'_r$ for each $r \in \mathcal{P}$.
  - $\chi((q,p)) = m \cdot \chi'((q,p))$ and for every channel $c \neq (q,p)$, $\chi'(c) = \chi(c)$.

  Then $(s, \chi) \stackrel{p?q}{\Longrightarrow} (s', \chi')$ and $(s', \chi') \in Conf_\mathcal{A}$.

Let $u \in \Sigma^*$. A run of $\mathcal{A}$ on u is a map $\rho : Pre(u) \to Conf_\mathcal{A}$ (where $Pre(u)$ is the set of prefixes of u) such that $\rho(\varepsilon) = (s_{in}, \chi_\varepsilon)$ and for each $\tau a \in Pre(u)$, $\rho(\tau) \stackrel{a}{\Longrightarrow} \rho(\tau a)$. The run ρ is *accepting* if $\rho(u)$ is a final configuration. Let $L(\mathcal{A}) = \{u \mid \mathcal{A}$ has an accepting run on $u\}$. It is easy to see that every member of $L(\mathcal{A})$ is complete and $L(\mathcal{A})$ is $\sim$-closed—that is, $u \in L(\mathcal{A})$ and $u \sim u'$ implies $u' \in L(\mathcal{A})$.

Unfortunately, $L(\mathcal{A})$ need not be regular. Consider, for instance, a message-passing automaton for the canonical producer-consumer system in which the producer p sends an arbitrary number of messages to the consumer q. Since we can reorder all the $p!q$ actions to be performed before all the $q?p$ actions, the queue in channel (p,q) is unbounded. Hence, the reachable configurations of this system are not bounded and the corresponding language is not regular.

For $B \in \mathbb{N}$, we say that a configuration (s, χ) of the message-passing automaton $\mathcal{A}$ is B*-bounded* if for every channel $c \in Chan$, it is the case that $|\chi(c)| \leq B$. We say that $\mathcal{A}$ is a B-bounded automaton if every reachable configuration $(s, \chi) \in Conf_\mathcal{A}$ is B-bounded.

Proposition 3.1. *Let $\mathcal{A}$ be a B-bounded automaton over Σ. Then $L(\mathcal{A})$ is a regular MSC language.*

This result follows easily from the definitions. Our goal is to prove the converse, which may be stated as follows.

Theorem 3.2. *Let L be a regular MSC language over Σ. Then, there is a* deterministic *B-bounded message-passing automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L$.*

Our strategy to prove this result is as follows. For a regular MSC language L, we consider the minimal DFA $\mathcal{A}_L$ for L. We construct a message-passing automaton $\mathcal{A}$ which simulates the behaviour of $\mathcal{A}_L$ on each complete word $u \in \Sigma^*$. The catch is that no single component of $\mathcal{A}$ is guaranteed to see all of u.

Thus, from the partial information available in each component about u, we have to reconstruct the behaviour of $\mathcal{A}_L$ on all of u. To achieve this, we need to time-stamp events so that components can keep track of each others' information about the computation.

4 Bounded time-stamps

Partial computations Let $u \in \Sigma^*$ be proper. A set of events $I \subseteq E_u$ is called an *(order) ideal* if I is closed with respect to $\leq$—that is, $e \in I$ and $f \leq e$ implies $f \in I$ as well.

Ideals denote consistent partial computations of u—notice that any linearization of an ideal forms a proper communication sequence.

p-views For an ideal I, the $\leq$-maximum p-event in I is denoted $\max_p(I)$, provided $\#_{\Sigma_p}(I) > 0$. The p-view of I, $\partial_p(I)$, is the ideal $\max_p(I)\downarrow$. Thus, $\partial_p(I)$ consists of all events in I which p can "see". (By convention, if $\max_p(I)$ is undefined—that is, if there is no p-event in I—the p-view $\partial_p(I)$ is empty.) For $P \subseteq \mathcal{P}$, we use $\partial_P(I)$ to denote $\bigcup_{p\in P} \partial_p(I)$.

Latest information Let $I \subseteq E_u$ be an ideal and $p, q \in \mathcal{P}$. Then $\mathit{latest}(I)$ denotes the set of events $\{\max_p(I) \mid p \in \mathcal{P}\}$. For $p \in \mathcal{P}$, we let $\mathit{latest}_p(I)$ denote the set $\mathit{latest}(\partial_p(I))$. A typical event in $\mathit{latest}_p(I)$ is of the form $\max_q(\partial_p(I))$ and denotes the $\leq$-maximum q-event in $\partial_p(I)$. This is the latest q-event in I that p knows about. For convenience, we denote this event $\mathit{latest}_{p\leftarrow q}(I)$. (As usual, if there is no q-event in $\partial_p(I)$, the quantity $\mathit{latest}_{p\leftarrow q}(I)$ is undefined.)

It is clear that for $q \neq p$, $\mathit{latest}_{p\leftarrow q}(I)$ will always correspond to a send action from Σ_q. However $\mathit{latest}_{p\leftarrow q}(I)$ need not be of the form $q!p$; the latest information that p has about q in I may have been obtained indirectly.

Message acknowledgments Let $I \subseteq E_u$ be an ideal and $e \in I$ an event of the form $p!q$. Then, e is said to have been *acknowledged* in I if the corresponding receive event f such that $e <_{pq} f$ exists and, moreover, belongs to $\partial_p(I)$. Otherwise, e is said to be *unacknowledged* in I.

Notice that it is not enough for a message to have been received in I to deem it to be acknowledged. We demand that the event corresponding to the receipt of the message be "visible" to the sending process.

For an ideal I and a pair of processes p, q, let $\mathit{unack}_{p\to q}(I)$ be the set of unacknowledged $p!q$ events in I.

B-bounded computations Let $u \in \Sigma^*$ be proper and let $M_u = (E_u, \leq, \lambda)$. We say that u is B-bounded, for $B \in \mathbb{N}$, if for every pair of processes p, q and for every ideal $I \subseteq E$, $\mathit{unack}_{p\to q}(I)$ contains at most B events.

The following result is immediate.

Proposition 4.1. *Let $u \in \Sigma^*$ be proper. The word u is B-bounded iff for every linearization v of M_u, for every prefix w of v and for every pair of processes p, q, $\#_{p!q}(w) - \#_{q?p}(w) \leq B$.*

It is easy to see that during the course of a B-bounded computation, none of the message buffers ever contains more than B undelivered messages, regardless of how the events are sequentialized. Thus, if each component $\mathcal{A}_p$ of a message-passing automaton is able to keep track of the sets $\{unack_{p\to q}(E_u)\}_{q\in\mathcal{P}}$ for each word u, this information can be used to inhibit sending messages along channels which are potentially saturated. This would provide a mechanism for constraining an arbitrary message-passing automaton to be B-bounded.

Primary information Let $I \subseteq E$ be an ideal. The *primary information* of I, $primary(I)$, consists of the following events in I:

- The set $latest(I) = \{\max_p(I) \mid p \in \mathcal{P}\}$.
- The collection of sets $unack(I) = \{unack_{p\to q}(I) \mid p, q \in \mathcal{P}\}$.

For $p \in \mathcal{P}$, we denote $primary(\partial_p(I))$ by $primary_p(I)$. Thus, $primary_p(I)$ reflects the primary information of p in I. Observe that for B-bounded computations, the number of events in $primary(I)$ is bounded.

In [13], a protocol is presented for processes to keep track of their primary information during the course of an arbitrary computation.[2] This protocol involves appending a bounded amount of information to each message in the underlying computation, provided the computation is B-bounded. To ensure that the message overhead is bounded, the processes use a distributed time-stamping mechanism which consistently assigns "names" to events using a bounded set of labels.

Consistent time-stamping Let $\mathcal{L}$ be a finite set of labels. For a proper communication sequence u, we say that $\tau : E_u \to \mathcal{L}$ is a *consistent time-stamping* of E_u by $\mathcal{L}$ if for each pair of (not necessarily distinct) processes p, q and for each ideal I the following holds: if $e_p \in primary_p(I)$, $e_q \in primary_q(I)$ and $\tau(e_p) = \tau(e_q)$ then $e_p = e_q$.

In the protocol of [13], whenever a process p sends a message to q, it first assigns a time-stamp to the new message from a finite set of labels. Process p then appends its primary information to the message being sent. Notice that the current send event will form part of the primary information since it is the latest p-event in $\partial_p(E_u)$. When q receives the message, it can consistently update its primary information to reflect the new information received from p.

The two tricky points in the protocol are for p to decide when it is safe to reuse a time-stamp, and for q to decide whether the information received from p is really new. In order to solve these problems, the protocol of [13] requires processes to also maintain additional time-stamps, corresponding to secondary information. Though we do not need the details of how the protocol works, we will need to refer to secondary information in the proof of our main theorem.

Secondary information Let I be an ideal. The *secondary information* of I is the collection of sets $primary(e\!\downarrow)$ for each event e in $primary(I)$. This collection

[2] In [13], the primary information of an ideal I is defined to include more events than just $latest(I) \cup unack(I)$. However, for our purposes, it suffices to treat events in $latest(I) \cup unack(I)$ as primary.

of sets is denoted $secondary(I)$. As usual, for $p \in \mathcal{P}$, $secondary_p(I)$ denotes the set $secondary(\partial_p(I))$.

In our framework, the protocol of [13] can now be described as follows.

Theorem 4.2. *For any $B \in \mathbb{N}$, we can construct a* deterministic *B-bounded message-passing automaton $\mathcal{A}^B = (\{\mathcal{A}_p^B\}_{p\in\mathcal{P}}, \mathcal{M}^B, s_{in}^B, \mathcal{F}^B)$ such that for every B-bounded proper communication sequence u, $\mathcal{A}^B$ inductively generates a consistent time-stamping τ of E_u. Moreover, for each component $\mathcal{A}_p^B$ of $\mathcal{A}^B$, the local state of $\mathcal{A}_p^B$ at the end of u records the information $primary_p(E_u)$ and $secondary_p(E_u)$ in terms of the time-stamps assigned by τ.*

5 Residues and decomposition

As we mentioned earlier, our strategy to prove our main theorem is to construct a message-passing automaton $\mathcal{A}$ which simulates the behaviour of the minimal DFA for L, $\mathcal{A}_L = (S, \Sigma, s_{in}, \delta, F)$, on each complete communication sequence u. In other words, after reading u, the components in $\mathcal{A}$ must be able to decide whether $\delta(s_{in}, u) \in F$. Unfortunately, after reading u each component in $\mathcal{A}$ only has partial information about $\delta(s_{in}, u)$—the component $\mathcal{A}_p$ only "knows about" those events from E_u which lie in the p-view $\partial_p(E_u)$. We have to devise a scheme to recover the state $\delta(s_{in}, u)$ from the partial information available with each process after reading u.

Another complication is that processes can only maintain a finite amount of information. We need a way of representing arbitrary words in a bounded, finite way. This can be done by recording for each word w, its "effect" as dictated by the minimal automaton $\mathcal{A}_L$. We associate with each word u a function $f_u : S \to S$, where S is the set of states of $\mathcal{A}_L$, such that $f_u(s) = \delta(s, u)$. The following observations follow from the fact that $\mathcal{A}_L$ is the minimal DFA recognizing L.

Proposition 5.1. *Let $u, w \in \Sigma^*$. Then:*

(i) $\delta(s_{in}, u) = f_u(s_{in})$.
(ii) $f_{uw} = f_w \circ f_u$, *where $\circ$ denotes function composition.*

Clearly the function $f_w : S \to S$ corresponding to a word w has a bounded representation. For an input u, if the components in $\mathcal{A}$ could compute the function f_u they would be able to determine whether $\delta(s_{in}, u) \in F$—by part (i) of the preceding proposition, $\delta(s_{in}, u) = f_u(s_{in})$. As the following result demonstrates, for any input u, it suffices to compute f_v for some linearization v of the MSC M_u.

Proposition 5.2. *Let $\hat{L}$ be a regular MSC language. For complete sequences $u, v \in \Sigma^*$, if $u \sim v$ then $f_u = f_v$.*

Proof: Follows from the structural properties of $\mathcal{A}_L$ described in Lemma 2.2. ☐

Before proceeding, we need a convention for representing the subsequence of communication actions generated by a subset of the events in an MSC.

Partial words Let $u = a_1 a_2 \ldots a_n$ be proper and let $X \subseteq E_u$ be given by $\{(i_1, a_{i_1}), (i_2, a_{i_2}), \ldots, (i_k, a_{i_k})\}$, where $i_1 < i_2 < \cdots < i_k$. Then, $u[X]$ denotes the subsequence $a_{i_1} a_{i_2} \ldots a_{i_k}$ (which need not be proper).

The following fact, analogous to standard results in Mazurkiewicz trace theory, will be used several times in our construction. We omit the proof.

Lemma 5.3. *Let u be proper and let $I, J \subseteq E_u$ be ideals such that $I \subseteq J$. Then $u[J] \sim u[I]u[J \setminus I]$.*

Corollary 5.4. *Let u be a word and $I_1 \subseteq I_2 \subseteq \cdots \subseteq I_k \subseteq E_u$ be a sequence of nested ideals. Then $u[I_k] \sim u[I_1]u[I_2 \setminus I_1] \cdots u[I_k \setminus I_{k-1}]$.*

Returning to our problem, suppose that $\mathcal{P}$ consists of m processes $\{p_1, p_2, \ldots, p_m\}$. Consider a complete word u. We wish to compute f_v for some $v \sim u$. Suppose we construct a chain of subsets of processes $\emptyset = Q_0 \subset Q_1 \subset Q_2 \subset \cdots \subset Q_m = \mathcal{P}$ such that for $j \in \{1, 2, \ldots, m\}$, $Q_j = Q_{j-1} \cup \{p_j\}$. From Corollary 5.4, we then have

$$u = u[\partial_{Q_m}(E_u)] \sim u[\partial_{Q_0}(E_u)]u[\partial_{Q_1}(E_u) \setminus \partial_{Q_0}(E_u)] \cdots u[\partial_{Q_m}(E_u) \setminus \partial_{Q_{m-1}}(E_u)]$$

Observe that $\partial_{Q_j}(E_u) \setminus \partial_{Q_{j-1}}(E_u)$ is the same as $\partial_{p_j}(E_u) \setminus \partial_{Q_{j-1}}(E_u)$. Thus, we can rewrite the expression above as

$$u = u[\partial_{Q_m}(E_u)] \sim u[\emptyset]u[\partial_{p_1}(E_u) \setminus \partial_{Q_0}(E_u)] \cdots u[\partial_{p_m}(E_u) \setminus \partial_{Q_{m-1}}(E_u)] \quad (\diamondsuit)$$

The word $u[\partial_{p_j}(E_u) \setminus \partial_{Q_{j-1}}(E_u)]$ is the portion of u which p_j has seen but which the processes in Q_{j-1} have not seen. This is a special case of what we call a residue.

Residues Let u be proper, $I \subseteq E_u$ an ideal and $p \in \mathcal{P}$ a process. $\mathcal{R}(u, p, I)$ denotes the word $u[\partial_p(E_u) \setminus I]$ and is called the *residue* of u at p with respect to I. Observe that any residue of the form $\mathcal{R}(u, p, I)$ can equivalently be written $\mathcal{R}(u, p, \partial_p(E_u) \cap I)$.

Using the notation of residues, we can write the word $u[\partial_{p_j}(E_u) \setminus \partial_{Q_{j-1}}(E_u)]$ as $\mathcal{R}(u, p_j, \partial_{Q_{j-1}}(E_u))$. A residue of this form is called a *process residue*: $\mathcal{R}(u, p, I)$ is a process residue if $\mathcal{R}(u, p, I) = \mathcal{R}(u, p, \partial_P(E_u))$ for some $P \subseteq \mathcal{P}$. We say that $\mathcal{R}(u, p, \partial_P(E_u))$ is the P-residue of u at p.

Unfortunately, a process residue at p may change due to an action of another process. For instance, if we extend a word u by an action $a = q?p$, it is clear that $\mathcal{R}(u, p, \partial_q(E_u))$ will not be the same as $\mathcal{R}(ua, p, \partial_q(E_{ua}))$ since q will get to know about more events from $\partial_p(u)$ after receiving the message via the action a. On the other hand, since p does not move on an action of the form $q?p$, p has no chance to update its q-residue when the action $q?p$ occurs.

However, it turns out that each process can maintain a set of residues based on its primary information such that these *primary residues* subsume the process residues. The key technical fact which makes this possible is the following.

Lemma 5.5. *For any non-empty ideal I, and $p, q \in \mathcal{P}$, the maximal events in $\partial_p(I) \cap \partial_q(I)$ lie in $primary_p(I) \cap primary_q(I)$.*

Proof: We show that for each maximal event e in $\partial_p(I) \cap \partial_q(I)$, either $e \in latest(\partial_p(I)) \cap unack(\partial_q(I))$ or $e \in unack(\partial_p(I)) \cap latest(\partial_q(I))$.

First suppose that $\partial_p(I) \setminus \partial_q(I)$ and $\partial_q(I) \setminus \partial_p(I)$ are both nonempty. Let e be a maximal event in $\partial_p(I) \cap \partial_q(I)$. Suppose e is an r-event, for some $r \in \mathcal{P}$. Since $\partial_p(I) \setminus \partial_q(I)$ and $\partial_q(I) \setminus \partial_p(I)$ are both nonempty, it follows that $r \notin \{p, q\}$. The event e must have $\leq$-successors in both $\partial_p(I)$ and $\partial_q(I)$. However, observe that any event f can have at most two immediate $\leq$-successors—one "internal" successor within the process and, if f is a send event, one "external" successor corresponding to the matching receive event.

Thus, the maximal event e must be a send event, with a $<_{rr}$ successor e_r and a $<_{rs}$ successor e_s, corresponding to some $s \in \mathcal{P}$. Assume that $e_r \in \partial_q(I) \setminus \partial_p(I)$ and $e_s \in \partial_p(I) \setminus \partial_q(I)$. Since the r-successor of e is outside $\partial_p(I)$, $e = \max_r(\partial_p(I))$. So e belongs to $latest(\partial_p(I))$. On the other hand, e is an unacknowledged $r!s$ event in $\partial_q(I)$. Thus, $e \in unack_{r \to s}(\partial_q(I))$, which is part of $unack(\partial_q(I))$.

Symmetrically, if $e_r \in \partial_p(I) \setminus \partial_q(I)$ and $e_s \in \partial_q(I) \setminus \partial_p(I)$, we find that e belongs to $unack(\partial_p(I)) \cap latest(\partial_q(I))$.

We still have to consider the case when $\partial_p(I) \subseteq \partial_q(I)$ or $\partial_q(I) \subseteq \partial_p(I)$. Suppose that $\partial_p(I) \subseteq \partial_q(I)$, so that $\partial_p(I) \cap \partial_q(I) = \partial_p(I)$. Let $e = \max_p(\partial_q(I))$. Clearly, $\partial_p(I) = e\!\downarrow$. Consider any r-event f in $\partial_p(I)$, where $r \notin \{p, q\}$. Since $f < e$, f cannot be maximal in $\partial_p(I)$. Thus, the only maximal event in $\partial_p(I)$ is the p-event e. Since e has a successor in $\partial_q(I)$, e must be a send event and is hence in $unack(\partial_p(I))$. Thus, $e \in unack(\partial_p(I)) \cap latest(\partial_q(I))$. Symmetrically, if $\partial_q(I) \subseteq \partial_p(I)$, the unique maximal event e in $\partial_q(I)$ belongs to $latest(\partial_p(I)) \cap unack(\partial_q(I))$. □

Let us call $\mathcal{R}(u, p, I)$ a *primary residue* if I is of the form $X\!\downarrow$ for some subset $X \subseteq primary_p(E_u)$. Clearly, for $p, q \in \mathcal{P}$, $\mathcal{R}(u, p, \partial_q(E_u))$, can be rewritten as $\mathcal{R}(u, p, \partial_p(E_u) \cap \partial_q(E_u))$. So, by the previous result the q-residue $\mathcal{R}(u, p, \partial_q(E_u))$ is a primary residue $\mathcal{R}(u, p, X\!\downarrow)$ for some $X \subseteq primary(\partial_p(E_u))$. Further, the set X can be effectively computed from the primary information of p and q. In fact, it turns out that *all* process residues can be effectively described in terms of primary residues.

We begin with a simple observation, whose proof we omit.

Proposition 5.6. *Let $u \in \Sigma^*$ be proper and $p \in \mathcal{P}$. For ideals $I, J \subseteq E_u$, let $\mathcal{R}(u,p,I)$ and $\mathcal{R}(u,p,J)$ be primary residues such that $\mathcal{R}(u,p,I) = \mathcal{R}(u,p,X_I\!\downarrow)$ and $\mathcal{R}(u,p,J) = \mathcal{R}(u,p,X_J\!\downarrow)$ for $X_I, X_J \subseteq primary_p(E_u)$. Then $\mathcal{R}(u,p,I \cup J)$ is also a primary residue and $\mathcal{R}(u,p,I \cup J) = \mathcal{R}(u,p,(X_I \cup X_J)\!\downarrow)$.*

Our claim that all process residues can be effectively described in terms of primary residues can then be formulated as follows.

Lemma 5.7. *Let $u \in \Sigma^*$ be proper, $p \in \mathcal{P}$ and $Q \subseteq \mathcal{P}$. Then $\mathcal{R}(u, p, \partial_Q(E_u))$ is a primary residue $\mathcal{R}(u, p, X{\downarrow})$ for p. Further, the set $X \subseteq primary_p(E_u)$ can be effectively computed from the information in $\bigcup_{q \in \{p\} \cup Q} primary_q(E_u)$.*

Proof: Let $Q = \{q_1, q_2, \ldots, q_k\}$. We can rewrite $\mathcal{R}(u, p, \partial_Q(E_u))$ as $\mathcal{R}(u, p, \bigcup_{i \in [1..k]} \partial_{q_i}(E_u))$. From Lemma 5.5 it follows that for each $i \in \{1, 2, \ldots, k\}$, p can compute a set $X_i \subseteq primary_p(E_u)$ from the information in $primary_p(E_u) \cup primary_{q_i}(E_u)$ such that $\mathcal{R}(u, p, \partial_{q_i}(E_u)) = \mathcal{R}(u, p, X_i{\downarrow})$. From Proposition 5.6, it then follows that $\mathcal{R}(u, p, \partial_Q(E_u)) = \mathcal{R}(u, p, \bigcup_{i \in \{1,2,\ldots,k\}} \partial_{q_i}(E_u)) = \mathcal{R}(u, p, X{\downarrow})$ where $X = \bigcup_{i \in \{1,2,\ldots,k\}} X_i$. ☐

6 Updating residues

We now describe how, while reading a word u, each process p maintains the functions f_w for each primary residue w of u at p.

Initially, at the empty word $u = \varepsilon$, every primary residue from $\{\mathcal{R}(u, p, X{\downarrow})\}_{p \in \mathcal{P}, X \subseteq primary(\partial_p(E_u))}$ is just the empty word ε. So, all primary residues are represented by the identity function $Id : \{s \mapsto s\}$.

Let $u \in \Sigma^*$ and $a \in \Sigma$. Assume inductively that every $p \in \mathcal{P}$ has computed at the end of u the function f_w for each primary residue $w = \mathcal{R}(u, p, X{\downarrow})$, where $X \subseteq primary(\partial_p(E_u))$. We want to compute for each p the corresponding functions after the word ua.

Suppose a is of the form $p!q$ and $X \subseteq primary_p(E_{ua})$. Let e_a denote the event corresponding to the new action a. If $e_a \in X$, then $\mathcal{R}(ua, p, X{\downarrow}) = \varepsilon$, so we represent the residue by the identity function Id. On the other hand, if $a \notin X$, then $X \subseteq primary_p(E_u)$, so we already have a residue of the form $\mathcal{R}(u, p, X)$. We then set $\mathcal{R}(ua, p, X{\downarrow})$ to be $f_a \circ \mathcal{R}(u, p, X{\downarrow})$. For $r \neq p$, the primary residues are unchanged when going from u to ua.

The case where a is of the form $p?q$ is more interesting. As before, the primary residues are unchanged for $r \neq p$. We show how to calculate all the new primary residues for p using the information obtained from q. This will use the following result.

Lemma 6.1. *Let $u \in \Sigma^*$ be proper. Let $p, q \in \mathcal{P}$ and $e \in E_u$ such that $e \in primary_q(E_u)$ but $e \notin \partial_p(E_u)$. Then $\mathcal{R}(u, p, e{\downarrow})$ is a primary residue $\mathcal{R}(u, p, X{\downarrow})$ for p. Further, the set $X \subseteq primary(\partial_p(E_u))$ can be effectively computed from the information in $primary_p(E_u)$ and $secondary_q(E_u)$.*

Proof: Let e be an r-event, $r \in \mathcal{P}$ and let $J = \partial_p(E_u) \cup e{\downarrow}$. By construction, $\max_p(J) = \max_p(E_u)$. On the other hand, $\max_r(J) = e$, since e is an r-event and we assumed that $e \notin \partial_p(E_u)$.

By Lemma 5.5, the maximal events in $\partial_p(J) \cap \partial_r(J)$ lie in $primary_p(J) \cap primary_r(J)$. Since $\max_p(J) = \max_p(E_u)$, $primary_p(J) = primary_p(E_u)$. On the other hand, $primary_r(J) = primary(e{\downarrow})$, which is a subset of $secondary_q(E_u)$, since $e \in primary_q(E_u)$.

Thus, the set of maximal events in $\partial_p(J) \cap \partial_r(J)$, which is the same as $\partial_p(E_u) \cap e\downarrow$, is contained in $\mathit{primary}_p(E_u) \cap \mathit{primary}(e\downarrow)$. These events are available in $\mathit{primary}_p(E_u) \cup \mathit{secondary}_q(E_u)$. $\square$

Suppose that $X \subseteq \mathit{primary}_p(E_{ua})$. Suppose that $X = \{x_1, x_2, \ldots, x_k\}$.

We first argue that for each $x_i \in X$, $\mathcal{R}(u, p, x_i\downarrow)$ is a primary residue $\mathcal{R}(u, p, Y_i\downarrow)$, where $Y_i \subseteq \mathit{primary}_p(E_u)$. If $x_i \in \mathit{primary}_p(E_u)$, then $\mathcal{R}(u, p, x_i\downarrow)$ is already a primary residue, so we can set $Y_i = \{x_i\}$. If, however, $x_i \notin \mathit{primary}_p(E_u)$, then x_i must have been contributed from $\mathit{primary}_q(u)$ through the message received at the action a. We have $x_i \in \mathit{primary}_q(E_u)$ but $x_i \notin \partial_p(E_u)$. Thus, appealing to Lemma 6.1, we can identify $Y_i \subseteq \mathit{primary}_p(E_u)$ such that $\mathcal{R}(u, p, \{x_i\}\downarrow) = \mathcal{R}(u, p, Y_i\downarrow)$.

Since $X = \bigcup_{i \in \{1,2,\ldots,k\}} x_i$, we can appeal to Proposition 5.6 to argue that $\mathcal{R}(u, p, X\downarrow)$ is the primary residue $\mathcal{R}(u, p, Y\downarrow)$ where $Y = \bigcup_{i \in \{1,2,\ldots,k\}} Y_i$. We can then set $\mathcal{R}(ua, p, X\downarrow) = f_a \circ \mathcal{R}(u, p, Y\downarrow)$.

Thus, after each action that is performed, the process performing the action can effectively update its primary residues using the primary and secondary information available to it.

7 A deterministic message-passing automaton for L

We can now construct a deterministic B-bounded message-passing automaton corresponding to a given regular MSC language L. We first observe that there is a bound $B \in \mathbb{N}$ such that every word in L is B-bounded.

Proposition 7.1. *Let $L \subseteq \Sigma^*$ be a regular MSC language. There is an effectively computable bound $B \in \mathbb{N}$ such that every word in L is B-bounded.*

Proof: Let $\mathcal{A}_L = (S, \Sigma, s_{in}, \delta, F)$ be the minimal DFA for L. Recall that we can associate with each live state in S and each channel $(p, q) \in \mathit{Chan}$ a channel-capacity function $\mathcal{K}_s : \mathit{Chan} \to \mathbb{N}$. Let B be the maximum value of $\mathcal{K}_s((p, q))$ over all live states s and all channels (p, q).

We know that for any word u in L, the run of $\mathcal{A}_L$ on u visits only live states. Thus, while processing u, no channel's capacity ever exceeds the bound B. Moreover, since L is $\sim$-closed, every interleaving v of M_u belongs to L and the run of $\mathcal{A}_L$ on each such interleaving also respects this bound. From Proposition 4.1, we can conclude that in every ideal $I \subseteq E_u$, for any pair p, q of processes, the set $\mathit{unack}_{p \to q}(I)$ contains at most B-events. Thus, u is B-bounded. $\square$

We now construct a B-bounded message-passing automaton $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in \mathcal{P}}, \mathcal{M}, s_{in}, \mathcal{F})$ for L, where B is the bound derived from the minimal DFA $\mathcal{A}_L$ for L as described in the preceding proposition.

Recall that $\mathcal{A}^B = (\{\mathcal{A}_p^B\}_{p \in \mathcal{P}}, \mathcal{M}^B, s_{in}^B, \mathcal{F}^B)$ is the time-stamping automaton for B-bounded computations, where the state of each component records the primary and secondary information of the component in terms of a consistent set of time-stamps.

– The message alphabet of $\mathcal{A}$ is the alphabet $\mathcal{M}^B$ used by the time-stamping automaton $\mathcal{A}^B$.
– In $\mathcal{A}$, a typical state of a component $\mathcal{A}_p$ is a pair (s_B, s_R) where s_B is a state drawn from $\mathcal{A}_p^B$ and s_R is the collection $\{f_X : S \to S\}_{X \subseteq primary_p(E_u)}$ of primary residues of $\mathcal{A}_p$ at the end of a word u.
– The local transition relation $\to_p$ of each component $\mathcal{A}_p$ is as follows:
  - For a of the form $p!q$, the tuple $((s_B, s_R), a, m, (s'_B, s'_R)) \in \to_p$ provided $(s_B, a, m, s'_B) \in \to_p^B$ and the residues in s'_R are derived from the residues in s_R using the time-stamping information in s_B, as described in Section 6.

    Moreover, according to the primary information in s_B, it should be the case that $|unack_{p\to q}(E_u)| < B$ for the word u read so far. Otherwise, this send action is disabled.
  - For a of the form $p?q$, the tuple $((s_B, s_R), a, m, (s'_B, s'_R)) \in \to_p$ provided $(s_B, a, m, s'_B) \in \to_p^B$ and the residues in s'_R are derived from the residues in s_R using the time-stamping information in s_B and the message m, as described in Section 6.
– In the initial state of $\mathcal{A}$, the local state of each component $\mathcal{A}_p$ is of the form $(s^p_{B,in}, s^p_{R,in})$ where $s^p_{B,in}$ is the initial state of $\mathcal{A}_p^B$ and $s^p_{R,in}$ records each residue to be the identity function Id.
– The global state $\{(s^p_B, s^p_R)\}_{p\in\mathcal{P}}$ belongs to the set $\mathcal{F}$ of final states if the primary residues stored in the global state record that $\delta(s_{in}, u) \in F$ for the word u read so far. (This is achieved by evaluating the expression ($\diamondsuit$) in Section 5.)

From the analysis of the previous section, it is clear that $\mathcal{A}$ accepts precisely the language L. The last clause in the transition relation $\to_p$ for send actions ensures that $\mathcal{A}$ will not admit a run in which $unack_{p\to q}(E_u)$ grows beyond B events for any input u and any pair of processes p, q. This ensures that every reachable configuration of $\mathcal{A}$ is B-bounded. Finally, we observe that $\mathcal{A}$ is deterministic because the time-stamping automaton $\mathcal{A}^B$ is deterministic and the update procedure for residues described in Section 6 is also deterministic.

We have thus succeeded in proving the main result we were after (Theorem 3.2)—namely, that for every regular MSC language L over Σ, there is a *deterministic* B-bounded message-passing automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L$.

We conclude by providing an upper bound for the size of $\mathcal{A}$, which can be computed by estimating the number of bits required to record the time-stamps and residues which form the local state of a process.

Proposition 7.2. *Let n be the number of processes in the system, m be the number of states of the minimal DFA $\mathcal{A}_L$ for L and B the bound computed from the channel-capacity functions of $\mathcal{A}_L$. Then, the number of local states of each component $\mathcal{A}_p$ is at most $2^{(2^{O(Bn^2)} m \log m)}$.*

References

1. Alur, R., Holzmann, G.J., and Peled, D.: An analyzer for message sequence charts. *Software Concepts and Tools*, **17(2)** (1996) 70–77.
2. Alur, R., and Yannakakis, M.: Model checking of message sequence charts. *Proc. CONCUR'99*, LNCS **1664**, Springer Verlag (1999) 114–129.
3. Ben-Abdallah, H., and Leue, S.: Syntactic detection of process divergence and non-local choice in message sequence charts. *Proc. TACAS'97*, LNCS **1217**, Springer-Verlag (1997) 259–274.
4. Booch, G., Jacobson, I., and Rumbaugh, J.: *Unified Modeling Language User Guide.* Addison Wesley (1997).
5. Damm, W., and Harel, D.: LCS's: Breathing life into message sequence charts. *Proc. FMOODS'99*, Kluwer Academic Publishers (1999) 293–312.
6. Diekert, V., and Rozenberg, G. (Eds.): *The book of traces.* World Scientific (1995).
7. Harel, D., and Gery, E.: Executable object modeling with statecharts. *IEEE Computer*, July 1997 (1997) 31–42.
8. Henriksen, J.G., Mukund, M., Narayan Kumar K., and Thiagarajan, P.S.: On message sequence graphs and finitely generated regular MSC languages, to appear in *Proc. ICALP 2000*, LNCS, Springer-Verlag (2000).
9. Henriksen, J.G., Mukund, M., Narayan Kumar K., and Thiagarajan, P.S.: Regular collections of message sequence charts, to appear in *Proc. MFCS 2000*, LNCS, Springer-Verlag (2000).
10. Ladkin, P.B., and Leue, S.: Interpreting message flow graphs. *Formal Aspects of Computing* **7**(5) (1975) 473–509.
11. Levin, V., and Peled, D.: Verification of message sequence charts via template matching. *Proc. TAPSOFT'97*, LNCS **1214**, Springer-Verlag (1997) 652–666.
12. Mauw, S., and Reniers, M.A.: High-level message sequence charts, *Proc. SDL '97*, Elsevier (1997) 291–306.
13. Mukund, M., Narayan Kumar, K., and Sohoni, M.: Keeping track of the latest gossip in message-passing systems. *Proc. Structures in Concurrency Theory (STRICT)*, Workshops in Computing Series, Springer-Verlag (1995) 249–263.
14. Muscholl, A.: Matching Specifications for Message Sequence Charts. *Proc. FOSSACS'99*, LNCS **1578**, Springer-Verlag (1999) 273–287.
15. Muscholl, A., Peled, D., and Su, Z.: Deciding properties for message sequence charts. *Proc. FOSSACS'98*, LNCS **1378**, Springer-Verlag (1998) 226–242.
16. Rudolph, E., Graubmann, P., and Grabowski, J.: Tutorial on message sequence charts. In *Computer Networks and ISDN Systems—SDL and MSC*, Volume 28 (1996).
17. Thomas, W.: Automata on infinite objects. In van Leeuwen, J. (Ed.), *Handbook of Theoretical Computer Science, Volume B*, North-Holland, Amsterdam (1990) 133–191.
18. Thomas, W.: Languages, Automata, and Logic. In Rozenberg, G., and Salomaa, A. (Eds.), *Handbook of Formal Language Theory, Vol. III*, Springer-Verlag, New York (1997) 389–455.
19. Zielonka, W.: Notes on finite asynchronous automata. *R.A.I.R.O.—Inf. Théor. et Appl.*, **21** (1987) 99–135.