A Theory of Regular MSC Languages

Jesper G. Henriksen [1]

BRICS, Computer Science Department, Aarhus University, Aarhus, Denmark [2]

Madhavan Mukund [*]

Chennai Mathematical Institute, Chennai, India

K. Narayan Kumar

Chennai Mathematical Institute, Chennai, India

Milind Sohoni

Indian Institute of Technology Bombay, Mumbai, India

P. S. Thiagarajan [3]

National University of Singapore, Singapore

Abstract

Message Sequence Charts (MSCs) are an attractive visual formalism widely used to capture system requirements during the early design stages in domains such as telecommunication software. It is fruitful to have mechanisms for specifying and reasoning about collections of MSCs so that errors can be detected even at the requirements level. We propose, accordingly, a notion of *regularity* for collections of MSCs and explore its basic properties. In particular, we provide an automata-theoretic characterization of regular MSC languages in terms of finite-state distributed automata called bounded message-passing automata. These automata consist of a set of sequential processes that communicate with each other by sending and receiving messages over bounded FIFO channels. We also provide a logical characterization in terms of a natural monadic second-order logic interpreted over MSCs.

A commonly used technique to generate a collection of MSCs is to use a Hierarchical Message Sequence Chart (HMSC). We show that the class of languages arising from the so-called *bounded* HMSCs constitute a proper subclass of the class of regular MSC languages. In fact, we characterize the bounded HMSC languages as the subclass of regular MSC languages that are *finitely generated*.

Key words: Message sequence charts, Message-passing systems, Regularity, Realizability, Synthesis, Monadic second-order logic

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 [19,31]. They also appear in the literature as sequence diagrams, message flow diagrams and object interaction diagrams and are used in a number of software engineering notational frameworks such SDL [31] and UML [7,14]. 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 the early stages of design. One question that naturally arises in this context is: What constitutes a reasonable collection of MSCs on which one can hope to do formal analysis? A related issue is how one should go about representing such collections.

In this paper, we propose *regular* collections of MSCs as the candidate for representing reasonable collections and present a variety of results in support of our proposal. We also present a number of representations of regular MSC collections and establish a strong connection to a standard way of representing MSC collections, namely, Hierarchical MSCs [25]. Preliminary versions of these results appeared in [17,18,26] where the notion of regular MSC languages and the related automata model were introduced.

Our notion of regularity has been guided by a number of concerns. The primary

* Corresponding Author. Address: Chennai Mathematical Institute, 92 G. N. Chetty Road, Chennai 600017, India. Fax: +91-44-28157671.

Email addresses: gulmann@brics.dk (Jesper G. Henriksen), madhavan@cmi.ac.in (Madhavan Mukund), kumar@cmi.ac.in (K. Narayan Kumar), sohoni@cse.iitb.ac.in (Milind Sohoni), thiagu@comp.nus.edu.sg (P. S. Thiagarajan).

[1] Present address: Airport Division, DSE A/S, Sverigesvej 19, DK-8700 Horsens, Denmark. E-mail: jgh@dse.dk

[2] BRICS: Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

[3] This work was partially supported by the NUS-SOC-ARF grant R-252-000-103-112.

one has been *finite-state realizability*. In other words, a good starting point for capturing the notion of a reasonable collection of MSCs is to demand that the behaviors denoted by the collection should be, as a whole, realizable by some finite-state device. A closely related concern is to synthesize systematically an executable specification—say in the form of an automaton—from a set of requirements as a regular collection of MSCs.

A standard way to generate a set of MSCs is to use a Hierarchical (or High-level) Message Sequence Chart (HMSC) [25]. An HMSC is a finite directed graph in which each node is itself labeled by an HMSC. The HMSCs that appear as the labels of the vertices may not refer to each other. Message Sequence Graphs (MSGs) are HMSCs in which each node is labeled by just an MSC (and not an HMSC). An MSG defines a collection of MSCs by concatenating the MSCs labeling each path from an initial vertex to a terminal vertex. Though HMSCs provide more succinct specifications than MSGs, they are only as expressive as MSGs. Thus, one often studies HMSCs in terms of MSGs [2,28,30].

In [2], Alur and Yannakakis investigate the restricted class of *bounded* (or *locally synchronized*) HMSCs. They show that the collection of MSCs generated by a bounded HMSC can be represented as a regular string language. As a result, the behaviors captured by a bounded HMSCs can be, in principle, realized as a finite-state automaton. It is easy to see that not every HMSC-definable collection of MSCs is realizable in this sense.

The main goal of this paper is to pin down this notion of realizability in terms of a notion of *regularity* for collections of MSCs and explore its basic properties. One consequence of our study is that our definition of regularity provides a general and robust setting for studying collections of MSCs which admits a number of different, but equivalent, representations. An important consequence is that our notion leads to a state-based representation that is one step closer to an implementation than the description in terms of MSCs. Stated differently, our work also addresses the issue, raised in [10], of converting inter-process descriptions at the level of requirements, as specified by MSCs, into intra-process executable specifications in terms of a reasonable model of computation.

Yet another motivation for focusing on regularity is that the classical notion of a regular collection of objects has turned out to be very fruitful in a variety of settings including finite (and infinite) strings, trees and restricted partial orders known as Mazurkiewicz traces [11,35,36]. In all these settings there is a representation of regular collections in terms of finite-state devices. There is also an accompanying monadic second-order logic that usually induces temporal logics using which one can reason about such collections [35]. One can then develop automated model-checking procedures for verifying properties specified in these temporal logics. In this context, the associated finite-state devices representing the regular collections often play a very useful role [37]. We show here that our notion of regular MSC languages fits in nicely with a related notion of a finite-state device, as also a monadic second-order logic.

In our study, we fix a finite set of processes $\mathcal{P}$ and consider $\mathcal{M}$, the universe of MSCs that the set $\mathcal{P}$ gives rise to. An MSC in $\mathcal{M}$ can be viewed as a labeled partial order in which the labels come from a finite alphabet Σ that is canonically fixed by $\mathcal{P}$. Our proposal for $L \subseteq \mathcal{M}$ to be regular is that the collection of all linearizations of all members of L should together constitute a regular subset of Σ^*. A crucial point is that, due to the communication mechanism of MSCs, the universe $\mathcal{M}$ itself is not a regular collection. This is in stark contrast to settings involving strings, trees or Mazurkiewicz traces. Futhermore, this distinction has a strong bearing on the automata-theoretic and logical formulations in our work. It turns out that regular MSC languages can be stratified using the concept of *bounds*. An MSC is said to be B-bounded if during any run of the MSC and at any stage in the run and for every pair of processes (p, q) there are at most B messages sent from p to q that have yet to be received by q. A language of MSCs is B-bounded if every member of the language is B-bounded. It turns out that every regular MSC language is B-bounded for some B. This leads to our automaton model called B-bounded message-passing automata. The components of such an automaton correspond to the processes in $\mathcal{P}$. The components communicate with each other over (potentially unbounded) FIFO channels. We say that a message-passing automaton is B-bounded if, during its operation, a channel never contains more than B messages. We establish a precise correspondence between B-bounded message-passing automata and B-bounded regular MSC languages. In a similar vein, we formulate a natural monadic second-order logic $\mathrm{MSO}(\mathcal{P}, B)$ interpreted over B-bounded MSCs. We then show that B-bounded regular MSC languages are exactly those that are definable in $\mathrm{MSO}(\mathcal{P}, B)$.

We also characterize exactly the regular MSC languages that can be represented by MSGs. In general, the MSC language defined by an MSG is not regular. Conversely, it turns out that there are regular MSC languages that can not be represented by an MSG. We show that the crucial link here is that of an MSC language being *finitely* generated. We prove that a regular MSC language can be represented by an MSG iff the language is finitely generated. As a by-product of this result we also show that a regular MSC language can be represented by an MSG iff it can be represented by a locally synchronized MSG.

As for related work, a number of studies are available that are concerned with individual MSCs in terms of their semantics and properties [1,21]. As pointed out earlier, a nice way to generate a collection of MSCs is to use an MSG. A variety of algorithms have been developed for MSGs in the literature—for instance, pattern matching [22,28,30] and detection of process divergence and non-local choice [5]. A systematic account of the various model-checking problems associated with MSGs and their complexities is given in [2]. The problem of model-checking MSGs with respect to formulas in Monadic Second-Order logic (MSO) is shown to be decidable in [23]. Note that the class of regular MSC languages and the class of MSG definable languages are incomparable. This decidability result has been extended to a generalization of MSGs called CMSGs (standing for Compositional MSGs) in [24]. The class of languages definable by CMSGs includes the class defined by MSGs as well as the class of regular MSC languages. The model-checking problem with respect to MSO is shown to be decidable for some infinite-state subclasses of HMSCs in [13]. For such subclasses the authors also show that equivalent communicating finite-state automata can be synthesised.

Recently, a new notion called *weak realizability* has been introduced in [3,4]. In this work, the target automata are message-passing automata (as we use them in this paper) with *local* rather than *global* accepting states. In the setting of Mazurkiewicz traces it is known that distributed automata with global acceptance conditions are strictly stronger than those with local acceptance conditions [38]. Trace languages accepted by automata with local accepting states are called *product languages* and are well-understood [33]. It would be interesting to extend the work of [3,4] to develop a corresponding theory of *product MSC languages*.

In this paper we confine our attention to *finite* MSCs and further we assume that each channel exhibits FIFO behaviour. As the recent results of [20,6] bear out, our results and techniques serve as a good launching pad for a similar account concerning infinite MSCs as well as to settings where messages may be delivered out of order.

The paper is structured as follows. In the next section we introduce MSCs and regular MSC languages. In Section 3 we establish our automata-theoretic characterization and, in Section 4, the logical characterization. While doing so, we borrow a couple of proof techniques from the theory of Mazurkiewicz traces [11]. However, we need to modify these techniques in a non-trivial way (especially in the setting of automata) due to the asymmetric flow of information via messages in the MSC setting, as opposed to the symmetric information flow via handshake communication in the trace setting.

We define Message Sequence Graphs in Section 5. We survey the existing body of theory for this class of labeled graphs and bring out the notion of locally synchronized MSGs. In Section 6 we define finitely generated languages and provide an effective procedure to decide whether a regular MSC language is finitely generated. Following this, we establish our characterization result for regular MSC languages that are MSG-representable.

2 Regular MSC Languages

Our study of regular MSC languages will focus on the most basic kind of MSCs—those that model communication through message-passing via reliable FIFOs. We ignore the actual content of the messages exchanged by the processes as well as internal events. Our aim is to clearly bring out the basic issues in the theory with as little clutter as possible. The theory that we develop will go through—with considerable notational overhead—in the presence of additional features such as handshake communication, non-FIFO channels, hierarchically structured states etc.

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite set of processes (agents) that communicate with each other through messages via reliable FIFO channels. For each $p \in \mathcal{P}$ we define $\Sigma_p \stackrel{\text{def}}{=} \{p!q \mid p \neq q\} \cup \{p?q \mid p \neq q\}$ 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. As mentioned above, at our level of abstraction, we shall not be concerned with the actual messages that are sent and received and we will also not deal with the internal actions of the agents. We set $\Sigma_{\mathcal{P}} = \bigcup_{p \in \mathcal{P}} \Sigma_p$ and let a, b range over $\Sigma_{\mathcal{P}}$. We also denote the set of *channels* by $Ch = \{(p, q) \mid p \neq q\}$ and let c, d range over Ch. Whenever the set of processes $\mathcal{P}$ is clear from the context, we will often write Σ instead of $\Sigma_{\mathcal{P}}$ etc.

Labelled posets A Σ-labelled poset is a structure $M = (E, \leq, \lambda)$ where $(E, \leq)$ is a poset and $\lambda : E \to \Sigma$ is a labelling function. For $e \in E$ we define $\downarrow e \stackrel{\text{def}}{=} \{e' \mid e' \leq e\}$. For $p \in \mathcal{P}$ and $a \in \Sigma$, we set $E_p \stackrel{\text{def}}{=} \{e \mid \lambda(e) \in \Sigma_p\}$ and $E_a \stackrel{\text{def}}{=} \{e \mid \lambda(e) = a\}$, respectively. For each $(p, q) \in Ch$, we define the relation $<_{pq}$ as follows:

$$e <_{pq} e' \iff \lambda(e) = p!q,\ \lambda(e') = q?p \text{ and } |{\downarrow e} \cap E_{p!q}| = |{\downarrow e'} \cap E_{q?p}|$$

Since messages are assumed to be read in FIFO fashion, $e <_{pq} e'$ implies that the message read by q at the receive event e' is the one sent by p at the send event e. Finally, for each $p \in \mathcal{P}$, we define the relation $\leq_{pp} \stackrel{\text{def}}{=} (E_p \times E_p) \cap \leq$, with $<_{pp}$ standing for the largest irreflexive subset of $\leq_{pp}$.

Definition 2.1 An MSC (over $\mathcal{P}$) is a *finite* Σ-labelled poset $M = (E, \leq, \lambda)$ that satisfies the following conditions[4]:

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

[4] Our definition captures the standard partial-order semantics associated with MSCs in, for instance, [1,31].

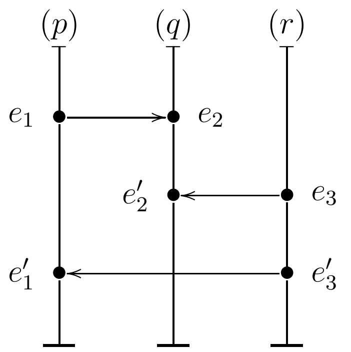

Fig. 1. An MSC over $\{p, q, r\}$.

(2) If $p \neq q$ then $|E_{p!q}| = |E_{q?p}|$.

(3) The partial order $\leq$ is the reflexive, transitive closure of the relation $\bigcup_{p,q \in \mathcal{P}} <_{pq}$.

□

In diagrams, the events of an MSC are presented in *visual order*. The events of each process are arranged in a vertical line and the members of the relation $<_{pq}$ are displayed as horizontal or downward-sloping directed edges from the vertical line corresponding to p to the vertical line corresponding to q. We illustrate the idea with an example, depicted in Figure 1. Here $\mathcal{P} = \{p, q, r\}$. For $x \in \mathcal{P}$, the events in E_x are arranged along the line labelled (x) with earlier (relative to $\leq$) events appearing above later events. For any two processes p, q, the $<_{pq}$-edges are depicted by horizontal edges—for instance $e_3 \ <_{rq}\ e_2'$. The labelling function λ is easy to extract from the diagram—for example, $\lambda(e_3') = r!p$ and $\lambda(e_2) = q?p$.

MSC languages Henceforth, we will identify an MSC with its isomorphism class. We let $\mathcal{M}_{\mathcal{P}}$ be the set of MSCs over $\mathcal{P}$. An *MSC language* is a subset $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P}}$. As before, we shall often omit $\mathcal{P}$ and denote $\mathcal{M}_{\mathcal{P}}$ by $\mathcal{M}$.

We shall define regular MSC languages in terms of their linearizations. For an MSC $M = (E, \leq, \lambda)$, we let $\mathit{lin}(M) \stackrel{\text{def}}{=} \{\lambda(\pi) \mid \pi \text{ is a linearization of } (E, \leq)\}$. By abuse of notation, we have used λ to also denote the natural extension of λ to E^*. For an MSC language $\mathcal{L} \subseteq \mathcal{M}$, we set $\mathit{lin}(\mathcal{L}) = \bigcup\{\mathit{lin}(M) \mid M \in \mathcal{L}\}$. In this sense, the string $p!q\ r!q\ q?p\ q?r\ r!p\ p?r$ is one linearization of the MSC in Figure 1.

In the literature (e.g. [1,29,30]) one sometimes considers a more generous notion of linearization where two *adjacent* receive actions in a process corresponding to messages from *different* senders are deemed causally independent. For instance, $p!q\ r!q\ q?r\ q?p\ r!p\ p?r$ would also be a valid linearization of the MSC in Figure 1. This is called the *causal order* of the MSC (as opposed to the visual order). Our results go through with suitable modifications even in the presence of this more generous notion of linearization.

Proper and complete words The notions of proper and complete words will be very useful for relating MSCs to their linearizations. For a word w and a letter a, we let $\#_a(w)$ denote the number of times a appears in w. We say that $\sigma \in \Sigma^*$ is *proper* if for every prefix τ of σ and every pair p, q of processes $\#_{p!q}(\tau) \geq \#_{q?p}(\tau)$. We say that σ is *complete* if σ is proper and $\#_{p!q}(\sigma) = \#_{q?p}(\sigma)$ for every pair p, q of processes.

An independence relation on complete words Next we define a *context-sensitive* independence relation $I \subseteq \Sigma^* \times (\Sigma \times \Sigma)$ as follows. $(\sigma, a, b) \in I$ iff the following conditions are satisfied :

- σab is proper
- $a \in \Sigma_p$ and $b \in \Sigma_q$ for distinct processes p and q
- $a = p!q$ and $b = q?p$ implies $\#_a(\sigma) > \#_b(\sigma)$.

We note that if $(\sigma, a, b) \in I$ then $(\sigma, b, a) \in I$.

We now set $\Sigma^\circ = \{\sigma \mid \sigma \in \Sigma^* \text{ and } \sigma \text{ is complete}\}$. Next we define $\sim \subseteq \Sigma^\circ \times \Sigma^\circ$ to be the least equivalence relation such that if $\sigma = \sigma_1 ab \sigma_2$, $\sigma' = \sigma_1 ba \sigma_2$ and $(\sigma_1, a, b) \in I$ then $\sigma \sim \sigma'$. For a complete word σ, we let $[\sigma]_\sim$ denote the equivalence class of σ with respect to $\sim$. It is important to note that $\sim$ is defined over Σ° and not Σ^*. It is easy to verify that for each $M \in \mathcal{M}$, $lin(M)$ is a subset of Σ° and is in fact a $\sim$-equivalence class over Σ°.

String MSC languages

We define $L \subseteq \Sigma^*$ to be a *string MSC language* if there exists an MSC language $\mathcal{L} \subseteq \mathcal{M}$ such that $L = lin(\mathcal{L})$. It is easy to see that $L \subseteq \Sigma^*$ is a string MSC language iff every string in L is complete and L is $\sim$-closed ; that is, if $\sigma \in L$ and $\sigma \sim \sigma'$ then $\sigma' \in L$.

Just as a Mazurkiewicz trace can be identified with its linearizations [11], we can identify each MSC with its linearizations. To formalize this, we construct representation maps $\mathsf{sm} : \Sigma^\circ/\!\sim\, \to \mathcal{M}$ and $\mathsf{ms} : \mathcal{M} \to \Sigma^\circ/\!\sim$ and argue that these maps are "inverses" of each other.

From linearizations to MSCs . . . Let $\sigma \in \Sigma^\circ$. Then $\mathsf{sm}(\sigma) = (E_\sigma, \leq, \lambda)$ where

- $E_\sigma = \{\tau a \mid \tau a \in \mathrm{prf}(\sigma)\}$, where $\mathrm{prf}(\sigma)$ is the set of prefixes of σ. In other words, $E_\sigma = \mathrm{prf}(\sigma) - \{\varepsilon\}$.
- $\leq\, = (R_{\mathcal{P}} \cup R_{Ch})^*$ where $R_{\mathcal{P}} = \bigcup_{p \in P} R_p$ and $R_{Ch} = \bigcup_{p,q \in P} R_{pq}$. The con-

stituent relations are defined as follows. For each $p \in \mathcal{P}$, $(\tau a, \tau' b) \in R_p$ iff $a, b \in \Sigma_p$ and $\tau a \in \text{prf}(\tau' b)$. Further, for each $p, q \in \mathcal{P}$, $(\tau a, \tau' b) \in R_{pq}$ iff $a = p!q$ and $b = q?p$ for some $p, q \in \mathcal{P}$ and in addition, $\#_a(\tau a) = \#_b(\tau' b)$.

- For $\tau a \in E$, $\lambda(\tau a) = a$.

It is easy to see that $\mathsf{sm}(\sigma)$ is an MSC with $<_{pp}= R_p$ and $<_{pq}= R_{pq}$. One can show that $\sigma \sim \sigma'$ implies $\mathsf{sm}(\sigma) = \mathsf{sm}(\sigma')$. We can thus extend sm to a map $\mathsf{sm}' : \Sigma^\circ/{\sim} \to \mathcal{M}$ given by $\mathsf{sm}'([\sigma]_\sim) = \mathsf{sm}(\sigma)$. Henceforth, we shall write sm to denote both sm and sm'.

. . . and back Conversely, we define the map $\mathsf{ms} : \mathcal{M} \to \Sigma^\circ/{\sim}$ by $\mathsf{ms}(M) = \mathit{lin}(M)$. It is easy to show that ms is well-defined. We can also show that for every $\sigma \in \Sigma^\circ, \mathsf{ms}(\mathsf{sm}(\sigma)) = [\sigma]_\sim$ and for every $M \in \mathcal{M}$, $\mathsf{sm}(\mathsf{ms}(M)) = M$. Thus $\Sigma^\circ/{\sim}$ and $\mathcal{M}$ are two equivalent ways of representing the same class of objects. Hence, abusing terminology, we will often write "MSC language" to mean "string MSC language". From the context, it should be clear whether we are working with labelled partial orders from $\mathcal{M}$ or complete strings over Σ^*. A good rule of thumb is that $\mathcal{L}$ will denote the former and L will denote the latter.

We can now finally define our notion of a regular collection of MSCs.

Definition 2.2 $\mathcal{L} \subseteq \mathcal{M}$ is a *regular MSC language* iff $\mathit{lin}(\mathcal{L})$ is a regular subset of Σ^*. □

Note that, unlike many standard settings (strings, trees or Mazurkiewicz traces), the universe $\mathcal{M}$ is itself *not* regular according to our definition because Σ° is not a regular subset of Σ^*. This fact has a strong bearing on the automata-theoretic and logical formulations of regular MSC languages as will become apparent in the later sections.

We now observe that there is an effective (and finitary) presentation of regular MSC languages.

Proposition 2.3 *It is decidable whether a regular subset $L \subseteq \Sigma^*$ is a regular MSC language.*

Proof:

Let $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$ be the minimal DFA representing L. We say that a state s of $\mathcal{A}$ is *live* if there is a path from s to a final state. It is not difficult to see that L is a regular MSC language iff we can associate with each live state $s \in S$, a channel-capacity function $\mathcal{K}_s : \mathit{Ch} \to \mathbb{N}$ that satisfies the following conditions.

(1) If $s \in \{s_{in}\} \cup\ F$ then $\mathcal{K}_s(c) = 0$ for every $c \in Ch$.
(2) 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 $c \neq (p, q)$.
(3) If s, s' are live states and $\delta(s, q?p) = s'$ 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 $c \neq (p, q)$.
(4) 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 $\delta(s_1', a) = s_2$.

□

Clearly the conditions enumerated in the proof can be checked in time linear in the size of the next state function δ.

We also point out that Item (4) in the proof above has useful consequences. By abuse of notation, let $\delta(s_{in}, u)$ denote the (unique) state reached by $\mathcal{A}$ on reading an input word u. Suppose u is a proper word and a, b are communication actions such that (u, a, b) belongs to the context-sensitive independence relation defined earlier. Then, due to Item (4), $\delta(s_{in}, uab) = \delta(s_{in}, uba)$. From this, we can conclude that if v, w are complete words such that $v \sim w$, then $\delta(s_{in}, v) = \delta(s_{in}, w)$.

We conclude this section by introducing the notion of B-bounded MSC languages. Let $B \in \mathbb{N}$ be a natural number. We say that a proper word σ is *weakly B-bounded* if for each prefix τ of σ and for each channel $(p, q) \in Ch$, $\#_{p!q}(\tau) - \#_{q?p}(\tau) \leq B$. We say that $L \subseteq \Sigma^\circ$ is weakly B-bounded if every word $\sigma \in L$ is weakly B-bounded.

Next we say the proper word σ is B-bounded if every w' with $w \sim w'$ is weakly B-bounded.

Turning now to MSCs, we shall say that the MSC M is B-bounded if every string in $lin(M)$ is weakly B-bounded. Since $lin(M)$ is an $\sim$-equivalence class, this is the same as saying that every string in $lin(M)$ is in fact B-bounded. Finally, a collection of MSCs is B-bounded if every member of the collection is B-bounded.

Proposition 2.4 *Let L be a regular MSC language. There is a bound $B \in \mathbb{N}$ such that L is B-bounded.*

Proof Sketch: From the proof of Proposition 2.3, it follows that every regular MSC language L is weakly B_L-bounded where the bound B_L is the largest value attained by the capacity functions attached to the live states in the minimal DFA for L. Since MSC languages are $\sim$-closed, it then follows that L is in fact B_L-bounded. $\Box$

3 An Automata-Theoretic Characterization

In what follows we assume the terminology and notation developed in the previous section. Recall that the set of processes $\mathcal{P}$ determines the communication alphabet Σ and that for $p \in \mathcal{P}$, Σ_p denotes the actions that process p participates in.

Definition 3.1 A *message-passing automaton* over Σ is a structure $\mathcal{A} = (\{\mathcal{A}_p\}_{p\in\mathcal{P}}, \Delta, s_{in}, F)$ where:

- Δ is a finite alphabet of messages.
- Each component $\mathcal{A}_p$ is of the form $(S_p, \longrightarrow_p)$ where
  - · S_p is a finite set of p-local states.
  - · $\longrightarrow_p \subseteq S_p \times \Sigma_p \times \Delta \times S_p$ 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.

$\Box$

The local transition relation $\longrightarrow_p$ 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 $\longrightarrow_p$ for each component $\mathcal{A}_p$ satisfies the following conditions:

- $(s, p!q, m_1, s'_1) \in \longrightarrow_p$ and $(s, p!q, m_2, s'_2) \in \longrightarrow_p$ imply $m_1 = m_2$ and $s'_1 = s'_2$.
- $(s, p?q, m, s'_1) \in \longrightarrow_p$ and $(s, p?q, m, s'_2) \in \longrightarrow_p$ imply $s'_1 = s'_2$.

In other words, 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.

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 $\chi : Ch \to \Delta^*$ is the *channel state* that 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 $F \times \{\chi_\varepsilon\}$.

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

- $(s_{in}, \chi_\varepsilon) \in Conf_{\mathcal{A}}$.
- Suppose $(s, \chi) \in Conf_{\mathcal{A}}$, (s', χ') is a configuration and $(s_p, p!q, m, s'_p) \in \longrightarrow_p$ 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 $(s_p, p?q, m, s'_p) \in \longrightarrow_p$ 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 $c \neq (q, p)$, $\chi'(c) = \chi(c)$.

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

Let $\sigma \in \Sigma^*$. A run of $\mathcal{A}$ over σ is a map $\rho : \mathrm{prf}(\sigma) \to Conf_{\mathcal{A}}$ such that $\rho(\varepsilon) = (s_{in}, \chi_\varepsilon)$ and for each $\tau a \in \mathrm{prf}(\sigma)$, $\rho(\tau) \stackrel{a}{\Longrightarrow} \rho(\tau a)$. The run ρ is *accepting* if $\rho(\sigma)$ is a final configuration. Note that a deterministic automaton has at most one run on any $\sigma \in \Sigma^*$.

We define $L(\mathcal{A}) \stackrel{\text{def}}{=} \{\sigma \mid \mathcal{A} \text{ has an accepting run over } \sigma\}$. It is easy to see that every member of $L(\mathcal{A})$ is complete and $L(\mathcal{A})$ is $\sim$-closed in the sense that if $\sigma \in L(\mathcal{A})$ and $\sigma \sim \sigma'$ then $\sigma' \in L(\mathcal{A})$. Consequently, $L(\mathcal{A})$ can be viewed as an MSC language.

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) can grow arbitrarily long. Hence, the set of reachable configurations of this system is 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 $c \in Ch$, it is the case that $|\chi(c)| \leq B$.

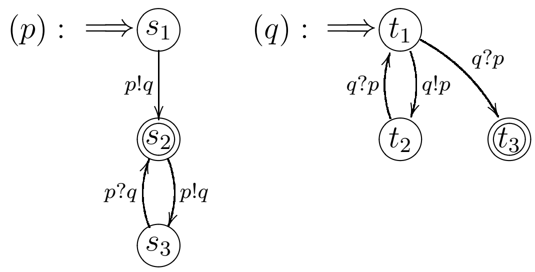

Fig. 2. A 3-bounded message-passing automaton.

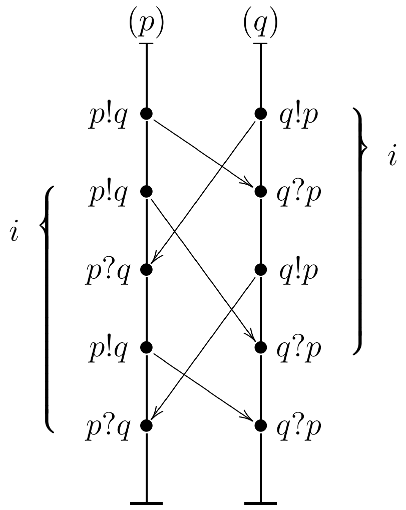

Fig. 3. The M_i's accepted by the automaton in Figure 2.

We say that $\mathcal{A}$ is a B-bounded automaton if every reachable configuration $(s, \chi) \in Conf_{\mathcal{A}}$ is B-bounded. It is not difficult to show that given a message-passing automaton $\mathcal{A}$ and a bound $B \in \mathbb{N}$, one can decide whether or not $\mathcal{A}$ is B-bounded. Figure 2 depicts an example of a 3-bounded message-passing automaton with two components, p and q. The initial state is (s_1, t_1) and there is only one final state, (s_2, t_3). (The message alphabet is a singleton and hence omitted.) The automaton accepts the infinite set of MSCs $\mathcal{L} = \{M_i\}_{i \in \mathbb{N}}$, where M_2 is displayed in Figure 3.

This automaton accepts an infinite set of MSCs, none of which can be expressed as the concatenation of two or more non-trivial MSCs. As a result, this MSC language cannot be represented using MSGs, as formulated in [2]. We will return to this point in Section 6.

The following result follows from the definitions. It constitutes the easy half of the characterization we wish to obtain.

Proposition 3.2 *Let $\mathcal{A}$ be a B-bounded message-passing automaton over Σ. Then $L(\mathcal{A})$ is a B-bounded regular MSC language.*

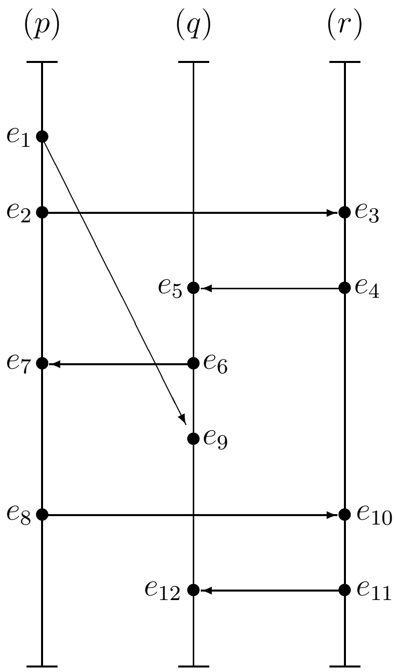

Fig. 4.

The second half of our characterization says that every B-bounded regular MSC language can be recognized by a B-bounded message-passing automaton. This is much harder to establish.

Let $L \subseteq \Sigma^*$ be a regular MSC language. As observed at the end of Section 2, the minimum DFA $\mathcal{A}_L$ for L yields a bound B such that L is B-bounded.

Our strategy to prove this result is as follows. For a regular MSC language L, we consider the minimum DFA $\mathcal{A}_L$ for L. We construct a message-passing automaton $\mathcal{A}$ that simulates the behaviour of $\mathcal{A}_L$ on each complete word $\sigma \in \Sigma^*$. The catch is that no single component of $\mathcal{A}$ is guaranteed to see all of σ. The partial information about σ that is available at each process can be formalized using ideals.

Ideals (prefixes) Let $\sigma \in \Sigma^*$ be proper. A set of events $I \subseteq E_\sigma$ 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 constitute consistent prefixes of σ—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 $\#_I(\Sigma_p) > 0$. The p-view of I, $\partial_p(I)$, is the ideal $\downarrow\max_p(I)$. Thus, $\partial_p(I)$ consists of all events in I that 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 $\bigcup_{p \in P} \partial_p(I)$.

Consider the MSC in Figure 4. The set of events $\{e_1, e_2, e_3, e_4, e_5, e_6, e_9\}$ form an ideal while the events $\{e_1, e_2, e_3, e_4, e_5, e_7\}$ do not.

Let I be the ideal $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$. The p-view of I is $\downarrow e_8 = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$. The q-view of I is $\downarrow e_9 = \{e_1, e_2, e_3, e_4, e_5, e_6, e_9\}$. The joint $\{p, q\}$-view of I is $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9\}$.

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

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

Proposition 3.3 *Let $\sigma, \rho \in \Sigma^*$. Then:*

(1) $\delta(s_{in}, \sigma) = f_\sigma(s_{in})$.
(2) $f_{\sigma\rho} = f_\rho \circ f_\sigma$, where $\circ$ denotes function composition.

Clearly the function $f_\sigma : S \to S$ corresponding to a word σ has a bounded representation. For an input σ, if the components in $\mathcal{A}$ could jointly compute the function f_σ they would be able to determine whether $\delta(s_{in}, \sigma) \in F$—by part (i) of the preceding proposition, $\delta(s_{in}, \sigma) = f_\sigma(s_{in})$. As the following result demonstrates, for any input σ, it suffices to compute f_ρ for some linearization ρ of the MSC $\mathsf{sm}(\sigma)$.

Proposition 3.4 *For complete sequences $\sigma, \rho \in \Sigma^*$, if $\sigma \sim \rho$ then $f_\sigma = f_\rho$.*

Proof: Follows from the structural properties of $\mathcal{A}_L$ described in Section 2. $\square$

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

Partial computations Let $\sigma = a_1 a_2 \dots a_n$ be proper and let $X \subseteq E_\sigma$ be given by $\{a_1 \dots a_{i_1}, a_1 \dots a_{i_2}, \dots, a_1 \dots a_{i_k}\}$, where $i_1 < i_2 < \cdots < i_k$. When we call X a *partial computation*, we mean that X should be identified with the induced labelled partial order $(E_X, \leq, \lambda)$ obtained by restricting E_σ to X. We denote by $\sigma[X]$ the set of linearizations of $(E_X, \leq, \lambda)$.

Observe that the linearizations of a partial computation are not, in general, proper words. Thus, if v and w are two linearizations of the same partial computation, it is quite likely that f_v and f_w are *not* the same function.

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

Lemma 3.5 *Let σ be proper and let $I, J \subseteq E_\sigma$ be ideals such that $I \subseteq J$. Then $\sigma[J] \supseteq \sigma[I]\sigma[J \setminus I]$.*

Corollary 3.6 *Let σ be a word and $I_1 \subseteq I_2 \subseteq \cdots \subseteq I_k \subseteq E_\sigma$ be a sequence of nested ideals. Then $\sigma[I_k] \supseteq \sigma[I_1]\sigma[I_2 \setminus I_1] \cdots \sigma[I_k \setminus I_{k-1}]$.*

3.1 Residues and decomposition

Returning to our problem of simulating the DFA $\mathcal{A}_L$ by a message-passing automaton, let $\mathcal{P}$ consist of m processes $\{p_1, p_2, \dots, p_m\}$. Consider a complete word σ. We wish to compute f_ρ for some $\rho \sim \sigma$. 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 $j \in \{1, 2, \dots, m\}$, $Q_j = Q_{j-1} \cup \{p_j\}$. From Corollary 3.6, we then have

$$\begin{aligned} [\sigma]_\sim &= \sigma[\partial_{Q_m}(E_\sigma)] \\ &\supseteq \sigma[\partial_{Q_0}(E_\sigma)]\sigma[\partial_{Q_1}(E_\sigma) \setminus \partial_{Q_0}(E_\sigma)] \cdots \sigma[\partial_{Q_m}(E_\sigma) \setminus \partial_{Q_{m-1}}(E_\sigma)] \end{aligned}$$

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

$$\begin{aligned} [\sigma]_\sim &= \sigma[\partial_{Q_m}(E_\sigma)] \\ &\supseteq \sigma[\emptyset]\sigma[\partial_{p_1}(E_\sigma) \setminus \partial_{Q_0}(E_\sigma)] \cdots \sigma[\partial_{p_m}(E_\sigma) \setminus \partial_{Q_{m-1}}(E_\sigma)] \end{aligned} \qquad (\diamondsuit)$$

Let us examine $(\diamondsuit)$ more closely. For each $i \in [1..m]$, let w_i be a linearization of the partial computation $\partial_{p_i}(E_\sigma) \setminus \partial_{Q_{i-1}}(E_\sigma)$. The expression $(\diamondsuit)$ then tells us that $\sigma \sim w_1 w_2 \dots w_m$.

Recall that different linearizations of a partial computation may give rise to different transition functions. However, $(\diamondsuit)$ tells us that we need not keep track of *all* linearizations of the partial computations $\partial_{p_i}(E_\sigma) \setminus \partial_{Q_{i-1}}(E_\sigma)$.

Suppose that each process p_i, $i \in [1..m]$, locally computes the function f_{w_i} corresponding to any one linearization w_i of the partial computation $\{\partial_{p_i}(E_\sigma) \setminus \partial_{Q_{i-1}}(E_\sigma)\}$. Then, from the global state at the end of the run, we can reconstruct f_σ by composing $f_{w_m} \circ f_{w_{m-1}} \circ \cdots \circ f_{w_1}$ to get $f_{w_1 w_2 \ldots w_m} = f_\sigma$. We can thus mark a global state as accepting if the composite function f_σ that it generates is such that $f_\sigma(s_{in}) \in F$.

In order to achieve this, each process p_j must inductively maintain information about the partial computation $\partial_{p_j}(E_\sigma) \setminus \partial_{Q_{j-1}}(E_\sigma)$. This partial computation represents the portion of σ that p_j has seen but the processes in Q_{j-1} have not seen. This is a special case of what we call a residue.

Residues Let σ be proper, $I \subseteq E_\sigma$ an ideal and $p \in \mathcal{P}$ a process. $\mathcal{R}(\sigma, p, I)$ denotes the set $\partial_p(E_\sigma) \setminus I$ and is called the *residue* of σ at p with respect to I. Observe that any residue of the form $\mathcal{R}(\sigma, p, I)$ can equivalently be written $\mathcal{R}(\sigma, p, \partial_p(E_\sigma) \cap I)$. Notice that a residue can be thought of as the induced labelled partial order defined by the events that it contains.

A residue of $\mathcal{R}(\sigma, p, I)$ is a process residue if $\mathcal{R}(\sigma, p, I) = \mathcal{R}(\sigma, p, \partial_P(E_\sigma))$ for some $P \subseteq \mathcal{P}$. We say that $\mathcal{R}(\sigma, p, \partial_P(E_\sigma))$ is the P-residue of σ at p.

Note that $\partial_{p_j}(E_\sigma) \setminus \partial_{Q_{j-1}}(E_\sigma)$ is a process residue. The expression ($\diamondsuit$) seems to suggest that each process should try and maintain information about linearizations of process residues locally. Unfortunately, a process residue at p may change due to an action of another process. For instance, if the word σ is extended by an action $a = q?p$, it is clear that $\mathcal{R}(\sigma, p, \partial_q(E_\sigma))$ will not be the same as $\mathcal{R}(\sigma a, p, \partial_q(E_{\sigma a}))$ since q will get to know about more events from $\partial_p(\sigma)$ 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.

Returning to the MSC in Figure 4, consider the proper word $\sigma = p!q\ p!r\ r?p\ r!q\ q?r\ q!p\ p?q\ p!q\ q?p$ corresponding to the (partial) linearization $e_1e_2e_3e_4e_5e_6e_7e_8e_9$. Let I denote the ideal corresponding to σ. Let J be the ideal $\{e_1, e_2, e_3, e_4, e_5\}$. The residue $\mathcal{R}(\sigma, p, J) = \{e_6, e_7, e_8\}$. This is *not* a process residue. The q-residue of σ at p, $\mathcal{R}(\sigma, q, \partial_q(I))$, is given by $\{e_7, e_8\}$. The r-residue of σ at p, $\mathcal{R}(\sigma, p, \partial_r(I))$, is given by $\{e_5, e_6, e_7, e_8\}$. However if we extend σ to $\sigma' = \sigma\ r?p$ generating the ideal $I' = I \cup \{e_{10}\}$, we find that $\mathcal{R}(\sigma', p, \partial_r(I')) = \emptyset$.

To get around this problem, each process will have to maintain residues in terms of local information that changes only when it moves. This information is called the *primary information* of a process. Maintaining and updating primary information requires a bounded time-stamping protocol, described in [27]. We now summarize the essential aspects of this protocol and then describe how to use it to fix the problem of maintaining process residues locally.

Recall that for a complete word σ, $\mathsf{sm}(\sigma) = (E_\sigma, \leq, \lambda)$ is the associated partial order defined on page 8. The map σ can be extended in a natural way to words that are proper but not complete. For such a proper word σ, the structure $(E_\sigma, \leq, \lambda)$ corresponds to an "incomplete" MSC in which some messages that have been sent have not yet been received. In fact, the resulting structure will be an ideal. In this sense, the correspondence between MSCs and complete words expressed by the maps sm and ms extends to a correspondence between ideals and proper words.

For the rest of this section, for any proper word σ, we implicitly use E_σ to denote the set of events associated with $\mathsf{sm}(\sigma)$.

Latest information Let $I \subseteq E_\sigma$ 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 $\mathit{latest}_{p \leftarrow q}(I)$. (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 $I \subseteq E_\sigma$ 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 $e <_{pq} f$ 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. The following proposition characterizes B-boundedness via the number of unacknowledged messages:

Consider the MSC in Figure 4. Let I be the ideal $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$. Then, $\mathit{latest}_{p \leftarrow q}(I) = e_6$. Notice that $\mathit{latest}_{q \leftarrow p}(I) = e_2$. This information about p comes to q via r and is more current than the direct message from p to q sent at e_1 that arrives at e_9.

In I, the message sent by p at e_2 is acknowledged (at e_7) while the message sent by p at e_1 is belongs to $unack_{p \to q}(I)$, though it has been received by q at e_9 within I.

Proposition 3.7 *Let $\sigma \in \Sigma^*$ be proper and let $\mathsf{sm}(\sigma) = (E_\sigma, \leq, \lambda)$. Then σ is B-bounded, for $B \in \mathbb{N}$, if and only if, for every pair of processes p, q and for every ideal $I \subseteq E_\sigma$, $unack_{p \to q}(I)$ contains at most B events.*

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 $\{unack_{p \to q}(E_\sigma)\}_{q \in \mathcal{P}}$ for each word σ, this information can be used to inhibit sending messages along channels that are potentially saturated and thus enforce B-boundedness. This would provide a mechanism for constraining an arbitrary message-passing automaton to be B-bounded.

Primary information Let $I \subseteq E_\sigma$ 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 [27], a protocol is presented for processes to keep track of their primary information during the course of an arbitrary computation. 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 that consistently assigns “names” to events using a bounded set of labels.

Consistent time-stamping Let Γ be a finite set of labels. For a proper communication sequence σ, we say that $\tau : E_\sigma \to \Gamma$ is a *consistent time-stamping* of E_σ by Γ 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 MSC shown in Figure 4, let I denote the entire collection of events $\{e_1, e_2, \ldots, e_{12}\}$. Here, $primary_p(I) = \{e_8, e_6, e_4\}$, $primary_q(I) = \{e_8, e_{12}, e_{11}\}$ and $primary_r(I) = \{e_8, e_6, e_{11}\}$. Thus, a consistent time-stamping is one that uses distinct labels for the events $\{e_6, e_8, e_{11}\}$ that span the primary information of more than one process in I. Normally, a consistent time-stamping would actually use distinct labels for all events that constitute primary information, namely $\{e_4, e_6, e_8, e_{11}, e_{12}\}$. Since $\{e_1, e_2\}$ are both send events that do not appear in the primary information of any process, a consistent time-stamping may assign both these events the same label as each other or, indeed, the same label as a third event.

In the protocol of [27], 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 $\partial_p(E_\sigma)$. 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 [27] 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 theorem.

Secondary information Let I be an ideal. The *secondary information* of I is the collection of sets $\mathit{primary}(\downarrow e)$ for each event e in $\mathit{primary}(I)$. This collection of sets is denoted $\mathit{secondary}(I)$. As usual, for $p \in \mathcal{P}$, $\mathit{secondary}_p(I)$ denotes the set $\mathit{secondary}(\partial_p(I))$.

In the MSC shown in Figure 4, let I denote the entire collection of events $\{e_1, e_2, \ldots, e_{12}\}$. Since $\mathit{primary}_p(I) = \{e_8, e_6, e_4\}$, $\mathit{secondary}_p(I) = \mathit{primary}(\downarrow e_8) \cup \mathit{primary}(\downarrow e_6) \cup \mathit{primary}(\downarrow e_4) = \{e_8, e_6, e_4\} \cup \{e_2, e_6, e_4\} \cup \{e_2, e_4\}$.

In our framework, the protocol of [27] can now be described as follows.

Theorem 3.8 *For any $B \in \mathbb{N}$, we can fix a set Γ of labels of size $O(B \times |\mathcal{P}|^2)$ and construct a* deterministic *B-bounded message-passing automaton $\mathcal{A}^B = (\{\mathcal{A}_p^B\}_{p \in \mathcal{P}}, \Delta^B, s_{in}^B, F^B)$ such that for every B-bounded proper communication sequence σ, $\mathcal{A}^B$ inductively generates a consistent time-stamping $\tau : E_\sigma \to \Gamma$. Moreover, for each component $\mathcal{A}_p^B$ of $\mathcal{A}^B$, the local state of $\mathcal{A}_p^B$ at the end of σ records the information $\mathit{primary}_p(E_\sigma)$ and $\mathit{secondary}_p(E_\sigma)$ in terms of the time-stamps assigned by τ.*

Actually, we need a more general version of this result, corresponding to maintaining consistent timestamps upto an arbitrary depth.

k-ary information Let I be an ideal. The *k-ary information* of I, $k\text{-}\mathit{ary}(I)$

is inductively defined as follows:

- $1\text{-}ary(I) = primary(I)$.
- For $k > 1$, $k\text{-}ary(I)$ is the collection of sets $primary(\downarrow e)$ for each event e in $(k{-}1)\text{-}ary(I)$.

As usual, for $p \in \mathcal{P}$, $k\text{-}ary_p(I)$ denotes the set $k\text{-}ary(\partial_p(I))$.

We can now extend the notion of a consistent time-stamping to arbitrary levels.

k-consistent time-stamping Let Γ be a finite set of labels and $k \in \mathbb{N}$. For a proper communication sequence σ, we say that $\tau : E_\sigma \to \Gamma$ is a *k-consistent time-stamping* of E_σ by Γ if for each pair of (not necessarily distinct) processes p, q and for each ideal I the following holds: if $e_p \in k\text{-}ary_p(I)$, $e_q \in k\text{-}ary_q(I)$ and $\tau(e_p) = \tau(e_q)$ then $e_p = e_q$.

In the MSC shown in Figure 4, let I denote the entire collection of events $\{e_1, e_2, \ldots, e_{12}\}$. The event e_2 is not a primary event but does lie within $secondary_p(I)$. Thus, a 2-consistent time-stamping would have to assign a distinct label to e_2, whereas a 1-consistent time-stamping can safely reuse the label assigned to e_2 within I.

The generalized version of Theorem 3.8 that we need is the following.

Theorem 3.9 *For any $B, k \in \mathbb{N}$, we can fix a set Γ of labels of size $O(B \times |\mathcal{P}|^{k+1})$ and construct a* deterministic *B-bounded message-passing automaton $\mathcal{A}^B = (\{\mathcal{A}_p^B\}_{p\in\mathcal{P}}, \Delta^B, s_{in}^B, F^B)$ such that for every B-bounded proper communication sequence σ, $\mathcal{A}^B$ inductively generates a k-consistent time-stamping $\tau : E_\sigma \to \Gamma$. Moreover, for each component $\mathcal{A}_p^B$ of $\mathcal{A}^B$, the local state of $\mathcal{A}_p^B$ at the end of σ records the information $k\text{-}\mathrm{ary}_p(E_\sigma)$ in terms of the time-stamps assigned by τ.*

3.3 Process and primary residues

With this background on primary information, we return to our problem of keeping track of residues. Recall that for a proper word σ, an ideal $I \subseteq E_\sigma$ and a process p, the residue $\mathcal{R}(\sigma, p, I)$ denotes the set $\partial_p(E_\sigma) \setminus I$. A residue $\mathcal{R}(\sigma, p, I)$ is a *process residue* for $P \subseteq \mathcal{P}$ if $I = \partial_P(E_\sigma)$. The goal is to maintain information about process residues locally at each process p, but the problem is that these residues may change even when p does not move, thereby making it impossible for p to directly represent this information.

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 that makes this possible is the following.

Lemma 3.10 *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 $\mathit{primary}_p(I) \cap \mathit{primary}_q(I)$.*

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

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 $\mathit{latest}(\partial_p(I))$. On the other hand, e is an unacknowledged $r!s$-event in $\partial_q(I)$. Thus, $e \in \mathit{unack}_{r \to s}(\partial_q(I))$, which is part of $\mathit{unack}(\partial_q(I))$.

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 $\mathit{unack}(\partial_p(I)) \cap \mathit{latest}(\partial_q(I))$.

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, $\partial_p(I) = {\downarrow} e$ and 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 $\mathit{unack}(\partial_p(I))$. Thus, $e \in \mathit{unack}(\partial_p(I)) \cap \mathit{latest}(\partial_q(I))$. Symmetrically, if $\partial_q(I) \subseteq \partial_p(I)$, the unique maximal event e in $\partial_q(I)$ belongs to $\mathit{latest}(\partial_p(I)) \cap \mathit{unack}(\partial_q(I))$. $\Box$

Let us call $\mathcal{R}(\sigma, p, I)$ a *primary residue* if I is of the form ${\downarrow} X$ for some subset $X \subseteq \mathit{primary}_p(E_\sigma)$. Clearly, for $p, q \in \mathcal{P}$, $\mathcal{R}(\sigma, p, \partial_q(E_\sigma))$, can be rewritten as $\mathcal{R}(\sigma, p, \partial_p(E_\sigma) \cap \partial_q(E_\sigma))$. From Lemma 3.10 it follows that the q-residue $\mathcal{R}(\sigma, p, \partial_q(E_\sigma))$ is a primary residue $\mathcal{R}(\sigma, p, {\downarrow} X)$ for some $X \subseteq \mathit{primary}(\partial_p(E_\sigma))$. Further, from the Lemma we know that 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 3.11 *Let $\sigma \in \Sigma^*$ be proper and $p \in \mathcal{P}$. For ideals $I, J \subseteq E_\sigma$, let $\mathcal{R}(\sigma, p, I)$ and $\mathcal{R}(\sigma, p, J)$ be primary residues such that $\mathcal{R}(\sigma, p, I) = \mathcal{R}(\sigma, p, \downarrow X_I)$ and $\mathcal{R}(\sigma, p, J) = \mathcal{R}(\sigma, p, \downarrow X_J)$ for $X_I, X_J \subseteq primary_p(E_\sigma)$. Then $\mathcal{R}(\sigma, p, I \cup J)$ is also a primary residue and $\mathcal{R}(\sigma, p, I \cup J) = \mathcal{R}(\sigma, p, \downarrow(X_I \cup X_J))$.*

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

Lemma 3.12 *Let $\sigma \in \Sigma^*$ be proper, $p \in \mathcal{P}$ and $Q \subseteq \mathcal{P}$. Then $\mathcal{R}(\sigma, p, \partial_Q(E_\sigma))$ is a primary residue $\mathcal{R}(\sigma, p, \downarrow X)$ for p. Further, the set $X \subseteq primary_p(E_\sigma)$ can be effectively computed from the information in $\bigcup_{q \in \{p\} \cup Q} primary_q(E_\sigma)$.*

Proof: Let $Q = \{q_1, q_2, \ldots, q_k\}$. We can rewrite $\mathcal{R}(\sigma, p, \partial_Q(E_\sigma))$ as $\mathcal{R}(\sigma, p, \bigcup_{i \in [1..k]} \partial_{q_i}(E_\sigma))$. From Lemma 3.10 it follows that for each $i \in \{1, 2, \ldots, k\}$, p can compute a set $X_i \subseteq primary_p(E_\sigma)$ from the information in $primary_p(E_\sigma) \cup primary_{q_i}(E_\sigma)$ such that $\mathcal{R}(\sigma, p, \partial_{q_i}(E_\sigma)) = \mathcal{R}(\sigma, p, \downarrow X_i)$. From Proposition 3.11, it then follows that $\mathcal{R}(\sigma, p, \partial_Q(E_\sigma)) = \mathcal{R}(\sigma, p, \bigcup_{i \in \{1,2,\ldots,k\}} \partial_{q_i}(E_\sigma)) = \mathcal{R}(\sigma, p, \downarrow X)$ where $X = \bigcup_{i \in \{1,2,\ldots,k\}} X_i$. □

3.4 Updating residues

Our strategy for constructing a message passing automaton for the regular MSC language L is to inductively have each process p maintain for each primary residue of the current input σ, the function f_w for some linearization w of the residue. Then, using the expression ($\diamondsuit$), the processes can jointly compute f_σ for the entire input σ.

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

Let $\sigma \in \Sigma^*$ be a proper word and let $a \in \Sigma$. Assume inductively that at the end of σ, for each $p \in \mathcal{P}$ and for every primary residue $\mathcal{R}(\sigma, p, \downarrow X)$ corresponding to $X \subseteq primary(\partial_p(E_\sigma))$, p has inductively computed f_w for some linearization w of $\mathcal{R}(\sigma, p, \downarrow X)$. We show how to update these functions for each process p after extending the computation from σ to σa.

Suppose a is of the form $p!q$ and $X \subseteq primary_p(E_{\sigma a})$. Let e_a denote the event corresponding to the new action a. If $e_a \in X$, then $\mathcal{R}(\sigma a, p, \downarrow X) = \varepsilon$, so we represent the residue by the identity function Id. On the other hand, if $e_a \notin X$, then $X \subseteq primary_p(E_\sigma)$, so we already have a function f_w corresponding to some linearization w of the residue $\mathcal{R}(\sigma, p, X)$. Since, every event in $\mathcal{R}(\sigma, p, X)$ is causally below the final a-event in $\mathsf{sm}(\sigma a)$, wa is a linearization of $\mathcal{R}(\sigma a, p, \downarrow X)$. Thus, we can extend f_w to $f_{wa} = f_a \circ f_w$.

For $r \neq p$, the primary residues are unchanged when going from σ to σa. We can thus extend the function f_w corresponding to the residue $\mathcal{R}(\sigma a, p, \downarrow X)$ to $f_{wa} = f_a \circ f_w$.

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

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

By Lemma 3.10, the maximal events in $\partial_p(J) \cap \partial_r(J)$ lie in $primary_p(J) \cap primary_r(J)$. Since $\max_p(J) = \max_p(E_\sigma)$, $primary_p(J) = primary_p(E_\sigma)$. On the other hand, $primary_r(J) = primary(\downarrow e)$, which is a subset of $secondary_q(E_\sigma)$, since $e \in primary_q(E_\sigma)$.

Thus, the set of maximal events in $\partial_p(J) \cap \partial_r(J)$, which is the same as $\partial_p(E_\sigma) \cap \downarrow e$, is contained in $primary_p(E_\sigma) \cap primary(\downarrow e)$. These events are available in $primary_p(E_\sigma) \cup secondary_q(E_\sigma)$. □

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

We first argue that for each $x_i \in X$, $\mathcal{R}(\sigma, p, \downarrow x_i)$ is a primary residue $\mathcal{R}(\sigma, p, \downarrow Y_i)$, where $Y_i \subseteq primary_p(E_\sigma)$. If $x_i \in primary_p(E_\sigma)$, then $\mathcal{R}(\sigma, p, \downarrow x_i)$ is already a primary residue, so we can set $Y_i = \{x_i\}$. If, however, $x_i \notin primary_p(E_\sigma)$, then x_i must have been contributed from $primary_q(\sigma)$ through the message received at the action a. We have $x_i \in primary_q(E_\sigma)$ but $x_i \notin \partial_p(E_\sigma)$. Thus, appealing to Lemma 3.13, we can identify $Y_i \subseteq primary_p(E_\sigma)$ such that $\mathcal{R}(\sigma, p, \downarrow\{x_i\}) = \mathcal{R}(\sigma, p, \downarrow Y_i)$.

Since $X = \bigcup_{i \in \{1,2,\ldots,k\}} x_i$, we can appeal to Proposition 3.11 to argue that $\mathcal{R}(\sigma, p, \downarrow X)$ is the primary residue $\mathcal{R}(\sigma, p, \downarrow Y)$ where $Y = \bigcup_{i \in \{1,2,\ldots,k\}} Y_i$. We can then set

$$\mathcal{R}(\sigma a, p, \downarrow X) = \mathcal{R}(\sigma, p, \downarrow Y) \cup \mathcal{R}(\sigma, q, \partial_p(E_\sigma) \cup \downarrow X_q) \cup \{e_a\}$$

where X_q is $X \cap primary_q(E_\sigma)$. Inductively, p maintains f_w for some linearization w of $\mathcal{R}(\sigma, p, \downarrow Y)$ and q communicates $f_{w'}$ for some linearization w' of $\mathcal{R}(\sigma, q, \partial_p(E_\sigma) \cup \downarrow X_q)$ in the current message to p. By construction, no event in $\mathcal{R}(\sigma, p, \downarrow Y)$ can be above any event in $\mathcal{R}(\sigma, q, \partial_p(E_\sigma) \cup \downarrow X_q)$ and both these sets of events lie below e_a. Thus, $w \circ w' \circ a$ is a valid linearization of $\mathcal{R}(\sigma a, p, \downarrow X)$ and p can compute the function $f_{ww'a} = f_a \circ f_{w'} \circ f_w$ from the information available to it after σ.

Thus, when each action is performed, the process performing the action can effectively update the functions corresponding to the linearizations of its primary residues using the primary and secondary information available to it.

3.5 A deterministic message-passing automaton for L

Let L be a regular MSC language and let B be the bound derived form the minimum DFA $\mathcal{A}_L$ for L as described earlier. We now construct a B-bounded message-passing automaton $\mathcal{A} = (\{\mathcal{A}_p\}_{p\in\mathcal{P}}, \Delta, s_{in}, F)$ for L.

Recall that $\mathcal{A}^B = (\{\mathcal{A}_p^B\}_{p\in\mathcal{P}}, \Delta^B, s_{in}^B, F^B)$ 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 Δ^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 $\{f_{w_X} : S \to S\}_{X \subseteq primary_p(E_\sigma)}$ of functions corresponding to one linearization w_X for each primary residue X of $\mathcal{A}_p$ at the end of a word σ.
- The local transition relation $\longrightarrow_p$ of each component $\mathcal{A}_p$ is as follows:
  - · For a of the form $p!q$, the tuple $((s_B, s_R), a, m, (s'_B, s'_R)) \in \longrightarrow_p$ provided $(s_B, a, m, s'_B) \in \longrightarrow_p^B$ and the functions in s'_R are derived from the functions in s_R using the time-stamping information in s_B, as described in Section 3.4.

    Moreover, according to the primary information in s_B, it should be the case that $|unack_{p\to q}(E_\sigma)| < B$ for the word σ read so far.
  - · For a of the form $p?q$, the tuple $((s_B, s_R), a, m, (s'_B, s'_R)) \in \longrightarrow_p$ provided $(s_B, a, m, s'_B) \in \longrightarrow_p^B$ and the functions in s'_R are derived from the functions in s_R using the time-stamping information in s_B and the message m, as described in Section 3.4.

- 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 $\mathcal{A}^B_p$ and $s^p_{R,in}$ records each function to be the identity function Id.
- The global state $\{(s^p_B, s^p_R)\}_{p\in\mathcal{P}}$ belongs to the set F of final states if the functions stored in the global state record that $\delta(s_{in}, \sigma) \in F$ for the word σ read so far. (This is achieved by evaluating the expression ($\Diamond$) in Section 3.1.)

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 $\longrightarrow_p$ for send actions ensures that $\mathcal{A}$ will not admit a run in which $unack_{p\rightarrow q}(E_\sigma)$ grows beyond B events for any input σ 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 3.4 is also deterministic. Thus, we have established the following:

Lemma 3.14 *Let $L \subseteq \Sigma^*$ be a B-bounded regular MSC language. Then there exists a B-bounded message-passing automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L$.*

The main result of this section, stated in the following theorem, is an easy consequence of the preceding Lemma combined with Proposition 3.2.

Theorem 3.15 *Let $L \subseteq \Sigma^*$. Then L is a regular MSC language if and only if there exists a 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 $\mathcal{A}$,

Proposition 3.16 *Let n be the number of processes in the system, m be the number of states of the minimum 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 $2^{(2^{O(Bn^2)} m \log m)}$.*

Proof: Each process p has to maintain one function from $S \rightarrow S$ for each subset of its primary events. The number of distinct primary events is bounded by $O(Bn^2)$—for any ideal I, there are at most $O(n)$ events in $latest_p(I)$ and $O(Bn^2)$ events in $unack_p(I)$. Thus, p has to maintain at most $2^{O(Bn^2)}$ functions from $S \rightarrow S$. Since each function from $S \rightarrow S$ can be written down using $m \log m$ bits, the entire state of p can be described using $2^{O(Bn^2)} m \log m$ bits, whence the result follows. □

4 A Logical Characterization

We formulate a monadic second-order logic that characterizes regular B-bounded MSC languages for each fixed $B \in \mathbb{N}$. Thus, our logic will be parameterized by a pair $(\mathcal{P}, B)$. For convenience, we fix $B \in \mathbb{N}$ through the rest of the section. As usual, we assume a supply of individual variables $x, y, \ldots$, a supply of set variables $X, Y, \ldots$, and a family of unary predicate symbols $\{Q_a\}_{a \in \Sigma}$. The syntax of the logic is then given by:

$$\mathrm{MSO}(\mathcal{P}, B) ::= Q_a(x) \mid x \in X \mid x \leq y \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid (\exists x)\varphi \mid (\exists X)\varphi.$$

Thus, the syntax does not reflect any information about the bound B or the structural features of an MSC. These aspects will be dealt with in the semantics. Let $\mathcal{M}(\mathcal{P}, B)$ be the set of B-bounded MSCs over $\mathcal{P}$. The formulas of our logic are interpreted over the members of $\mathcal{M}(\mathcal{P}, B)$. Let $M = (E, \leq, \lambda)$ be an MSC in $\mathcal{M}(\mathcal{P}, B)$ and $\mathcal{I}$ be an interpretation that assigns to each individual variable x a member $\mathcal{I}(x)$ in E and to each set variable X a subset $\mathcal{I}(X)$ of E. Then $M \models_{\mathcal{I}} \varphi$ denotes that M satisfies φ under $\mathcal{I}$. This notion is defined in the expected manner—for instance, $M \models_{\mathcal{I}} Q_a(x)$ if $\lambda(\mathcal{I}(x)) = a$, $M \models_{\mathcal{I}} x \leq y$ if $\mathcal{I}(x) \leq \mathcal{I}(y)$ etc. For convenience, we have used $\leq$ to denote both the predicate symbol in the logic and the corresponding causality relation in the model M.

As usual, φ is a sentence if there are no free occurrences of individual or set variables in φ. With each sentence φ we can associate an MSC language $\mathcal{L}_\varphi \stackrel{\text{def}}{=} \{M \in \mathcal{M}(\mathcal{P}, B) \mid M \models \varphi\}$. We say that $\mathcal{L} \subseteq \mathcal{M}(\mathcal{P}, B)$ is $\mathrm{MSO}(\mathcal{P}, B)$-definable if there exists a sentence φ such that $\mathcal{L}_\varphi = \mathcal{L}$. For convenience, we often use "definable" to mean "$\mathrm{MSO}(\mathcal{P}, B)$-definable". We wish to argue that $\mathcal{L} \subseteq \mathcal{M}(\mathcal{P}, B)$ is definable iff it is a B-bounded regular MSC language. It turns out that the techniques used for proving a similar result in the theory of traces [12] can be suitably modified to derive our result.

Lemma 4.1 *Let φ be a sentence in* $\mathrm{MSO}(\mathcal{P}, B)$*. Then $\mathcal{L}_\varphi$ is a B-bounded regular MSC language.*

Proof Sketch: The fact that $\mathcal{L}_\varphi$ is B-bounded follows from the semantics and hence we just need to establish regularity. Consider $\mathrm{MSO}(\Sigma)$, the monadic second-order theory of finite strings in Σ^*. This logic has the same syntax as $\mathrm{MSO}(\mathcal{P}, B)$ except that, to avoid confusion, we will use the predicate symbol $\preceq$ instead of $\leq$ and interpret $\preceq$ as the usual ordering relation over the positions of a structure in Σ^*. Let $L = \mathit{lin}(\mathcal{L}_\varphi)$. We exhibit a sentence $\widehat{\varphi}$ in $\mathrm{MSO}(\Sigma)$ such that $L = \{\sigma \mid \sigma \models \widehat{\varphi}\}$. The required conclusion will then follow from Büchi's theorem [8]. Let $\{\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_n\}$ be the set $\{\mathcal{K} \in \mathbb{N}^{Ch} \mid \forall c \in Ch.\ \mathcal{K}(c) \leq B\}$. Without loss of generality, assume that $\mathcal{K}_0(c) = 0$ for every $c \in Ch$. For $\mathcal{K} \in$

$\mathbb{N}^{Ch}$ and $c \in Ch$, let $\mathcal{K}^{++c}$ to be the member of $\mathbb{N}^{Ch}$ where $\mathcal{K}^{++c}(c) = \mathcal{K}(c)+1$ and $\mathcal{K}^{++c}(d) = \mathcal{K}(d)$ for all $d \neq c$. Similarly, for $\mathcal{K} \in \mathbb{N}^{Ch}$ and $c \in Ch$ such that $\mathcal{K}(c) > 0$, $\mathcal{K}^{--c}$ is given by $\mathcal{K}^{--c}(c) = \mathcal{K}(c) - 1$ and $\mathcal{K}^{--c}(d) = \mathcal{K}(d)$ for all $d \neq c$.

The required sentence $\widehat{\varphi}$ will be of the form:

$$(\exists X_{\mathcal{K}_0})(\exists X_{\mathcal{K}_1}) \cdots (\exists X_{\mathcal{K}_n})(\mathit{COMP} \wedge \mathit{B\text{-}BOUNDED} \wedge \|\varphi\|)$$

where COMP, $\mathit{B\text{-}BOUNDED}$, and $\|\varphi\|$ are defined as follows. We provide these definitions in textual form to enhance readability. They can be easily converted to formulas in MSO(Σ).

First we define COMP to be the conjunction of the following formulas.

(1) Every position x belongs to exactly one of the sets in $\{X_{\mathcal{K}_0}, \ldots, X_{\mathcal{K}_n}\}$.
(2) If x is the first position then $x \in X_{\mathcal{K}_0}$.
(3) If x is the last position then $Q_{q?p}(x)$ for some $c = (p,q)$. Moreover x belongs to $X_{\mathcal{K}_m}$ such that $\mathcal{K}_m(c) = 1$ and $\mathcal{K}_m(d) = 0$ for $d \neq c$.
(4) If y is the successor of x, $Q_{p!q}(x)$, $x \in X_{\mathcal{K}_i}$ and $y \in X_{\mathcal{K}_j}$, then $\mathcal{K}_j = \mathcal{K}_i^{++c}$, where $c = (p,q)$.
(5) If y is the successor of x, $Q_{q?p}(x)$, $x \in X_{\mathcal{K}_i}$ and $y \in X_{\mathcal{K}_j}$, then $\mathcal{K}_i(c) > 0$ and $\mathcal{K}_j = \mathcal{K}_i^{--c}$, where $c = (p,q)$.

The formula $\|\varphi\|$ is given inductively as follows:

- $\|Q_a(x)\| \stackrel{\text{def}}{=} Q_a(x)$.
- $\|x \in X\| \stackrel{\text{def}}{=} x \in X$.
- $\|\neg\varphi'\| \stackrel{\text{def}}{=} \neg\|\varphi'\|$.
- $\|\varphi_1 \vee \varphi_2\| \stackrel{\text{def}}{=} \|\varphi_1\| \vee \|\varphi_2\|$.
- $\|(\exists x)\varphi'\| \stackrel{\text{def}}{=} (\exists x)\|\varphi'\|$.
- $\|(\exists X)\varphi'\| \stackrel{\text{def}}{=} (\exists X)\|\varphi'\|$.
- Finally, $\|x \leq y\| \stackrel{\text{def}}{=} x \sqsubseteq y$ where we shall first define $\sqsubseteq$ in terms of $\sqsubset\!\cdot$ and then define $\sqsubset\!\cdot$. This translation is based on the fact that in an MSC $M = (E, \leq, \lambda)$, $\leq = (\bigcup_{p,q \in \mathcal{P}} <_{pq} \cup \bigcup_{p \in \mathcal{P}} \leq_{pp})^*$.

The formula $x \sqsubseteq y$ asserts existence of non-empty subsets $\{p_1, p_2, \ldots, p_m\}$ of processes and $\{x_1, y_1, x_2, y_2, \ldots, x_m, y_m\}$ of positions such that $x = x_1$ and $y_m = y$. Further, $x_i \preceq y_i$ and x_i and y_i are both in Σ_{p_i} for $1 \leq i \leq m$. In addition, $y_i \sqsubset\!\cdot\, x_{i+1}$ for $1 \leq i < m$.

The predicate $x \sqsubset\!\cdot\, y$ is given by: $x \prec y$ and there is a channel $c = (p,q)$ such that $Q_{p!q}(x)$ and $Q_{q?p}(y)$. Further, if $x \in X_{\mathcal{K}_m}$ then there are exactly $\mathcal{K}_m(c)$ occurrences of the symbol $q?p$ between the positions x and y (and not including y).

The formula *B-BOUNDED* asserts that the word under consideration is B-bounded. Proposition 3.7 implies that a word w violates B-boundedness if and only if there is a $q?p$-event that is causally independent of (i.e. incomparable under $\sqsubseteq$ from) at least $(B+1)$ $p!q$-events. This can be easily stated in MSO.

It is now straightforward to show that $\widehat{\varphi}$ has the required property.

$\square$

Lemma 4.2 *Let $\mathcal{L} \subseteq \mathcal{M}(\mathcal{P}, B)$ be a regular MSC language. Then $\mathcal{L}$ is definable in* MSO$(\mathcal{P}, B)$.

Let $L = \mathit{lin}(\mathcal{L})$. Then L is a regular (string) MSC language over Σ. Hence by Büchi's theorem [8] there exists a sentence φ in MSO(Σ) such that $L = \{\sigma \mid \sigma \models \varphi\}$. An important property of φ is that one linearization of an MSC satisfies φ iff all linearizations of the MSC satisfy φ. We then define the sentence $\widehat{\varphi} = \|\varphi\|$ in MSO$(\mathcal{P}, B)$ inductively such that the language of MSCs defined by $\widehat{\varphi}$ is precisely $\mathcal{L}$. The key idea here is to define a canonical linearization of MSCs and show that the underlying linear order is expressible in MSO$(\mathcal{P}, B)$. As a result, we can look for a formula $\widehat{\varphi}$ that will say "along the canonical linearization of an MSC, the sentence φ is satisfied". We present below the main ideas and constructions involved in arriving at $\widehat{\varphi}$.

Throughout what follows, we fix a strict linear order $\prec \subseteq \Sigma \times \Sigma$. Consider an MSC $M = (E, \leq, \lambda)$. For $e \in E$, let $\uparrow e = \{e' \mid e \leq e'\}$. For events $e, e' \in E$, we define $e \; co \; e'$ if $e \not\leq e'$ and $e' \not\leq e$. For $X \subseteq E$, let $\lambda(X) = \{\lambda(e) \mid e \in X\}$. Next, suppose that $\emptyset \neq \Sigma' \subseteq \Sigma$. Then $\min(\Sigma')$ is the least element of Σ' under $\prec$. Finally, suppose $e, e' \in E$ with $e \; co \; e'$. Then $\Sigma_{ee'} = \lambda(\uparrow e \setminus \uparrow e')$.

Let $M = (E, \leq, \lambda)$ be an MSC. Then the ordering relation $\prec$ induces the ordering relation $\prec_M \subseteq E \times E$ given by $e \prec_M e'$ if $e < e'$ or ($e \; co \; e'$ and $\min(\Sigma_{ee'}) \prec \min(\Sigma_{e'e})$).

Claim 4.3 *Let $M = (E, \leq, \lambda)$ be an MSC. Then $(E, \prec_M)$ is a strict linear order and $\prec_M$ is a linearization of $\leq$.*

Proof: Same as the proof of [34, Lemma 15], which asserts an identical result in the setting of (infinite) Mazurkiewicz traces. $\square$

We next exhibit a formula in MSO$(\mathcal{P}, B)$ (for any $B \in \mathbb{N}$) that captures the relation $\prec_M$ for each B-bounded MSC M. First we define the formula

$\min(z_1, z_2, a)$ where z_1 and z_2 are individual variables and $a \in \Sigma$ via:

$$\min(z_1, z_2, a) = (\exists z)[z_1 \leq z \wedge \neg(z_2 \leq z) \wedge Q_a(z) \wedge \\ (\forall z')\,((z_1 \leq z' \wedge \neg(z_2 \leq z')) \Rightarrow Q_a(z') \vee \textstyle\bigvee_{a \prec a'} Q_{a'}(z'))]$$

The formula $\mathrm{Lex}(x, y)$ is now given by:

$$\mathrm{Lex}(x, y) = (x < y) \vee \left(\mathrm{co}(x, y) \wedge \bigvee_{a \prec b} \min(x, y, a) \wedge \min(y, x, b)\right)$$

where $\mathrm{co}(x, y)$ is an abbreviation for $\neg(x \leq y) \wedge \neg(y \leq x)$.

Turning now to the proof of Lemma 4.2, let $L = \mathit{lin}(\mathcal{L})$. Then L is a regular (string) MSC language over Σ. Hence by Büchi's theorem [8] there exists a sentence φ in $\mathrm{MSO}(\Sigma)$ such that $L = \{\sigma \mid \sigma \models \varphi\}$. We now define the formula $\widehat{\varphi} = \|\varphi\|$ in $\mathrm{MSO}(\mathcal{P}, B)$ inductively as follows:

$$\|Q_a(x)\| = Q_a(x) \text{ and } \|x \preceq y\| = (x \leq y \wedge y \leq x) \vee \mathrm{Lex}(x, y)$$

The remaining clauses are the natural ones. It is now straightforward to verify that $\mathcal{L}_{\widehat{\varphi}} = \mathcal{L}$. The key step in the proof is to show the following: Suppose $M \in \mathcal{M}(\mathcal{P}, B)$ and σ is the linearization of M dictated by $\prec_M$. Then M is a model of $\widehat{\varphi}$ iff σ is a model of φ. This follows easily by structural induction on φ. The required conclusion can now be derived by exploiting the fact that L is $\sim$-closed.

Since $\mathrm{MSO}(\Sigma)$ is decidable, it follows that $\mathrm{MSO}(\mathcal{P}, B)$ is decidable as well. We can now summarize the results characterizing regularity as follows.

Theorem 4.4 *Let $L \subseteq \Sigma^*$, where Σ is the communication alphabet associated with a set $\mathcal{P}$ of processes. Then, the following are equivalent.*

(i) *L is a regular MSC language.*
(ii) *L is a B-bounded regular MSC language, for some $B \in \mathbb{N}$.*
(iii) *There exists a bounded message-passing automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L$.*
(iv) *L is $\mathrm{MSO}(\mathcal{P}, B)$-definable, for some $B \in \mathbb{N}$.*

5 Message Sequence Graphs

The standard method to describe multiple communication scenarios is to generate collections of MSCs by means of Hierarchical Message Sequence Charts (HMSCs). As described in the introduction, to analyze HMSCs, it suffices to

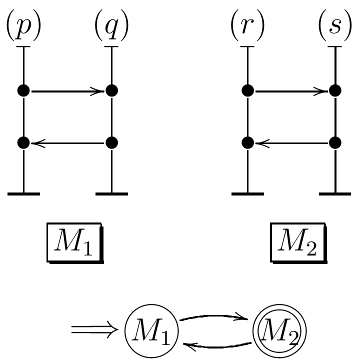

Fig. 5. An example MSG.

flatten them out to obtain Message Sequence Graphs (MSGs). As a consequence, henceforth we concentrate on MSGs rather than HMSCs.

An MSG allows the protocol designer to write a finite specification that combines MSCs using basic operations such as branching choice, composition and iteration. Such MSGs are finite directed graphs with designated initial and terminal vertices. Each vertex in an MSG is labelled by an MSC. The edges represent the natural operation of MSC concatenation. The collection of MSCs represented by an MSG consists of all those MSCs obtained by tracing a path in the MSG from an initial vertex to a terminal vertex and concatenating the MSCs that are encountered along the path.

Formally, the (asynchronous) concatenation of MSCs is defined as follows. 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 $\leq \, = (\bigcup_{p,q\in\mathcal{P}} <_{pq})^*$, where $<_{pp} = <^1_{pp} \cup <^2_{pp} \cup \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2, \lambda(e_1) \in \Sigma_p, \lambda(e_2) \in \Sigma_p\}$ and for $(p, q) \in Ch$, $<_{pq} = <^1_{pq} \cup <^2_{pq}$.

We can now formally define MSGs. A *Message Sequence Graph (MSG)* is a structure $\mathcal{G} = (Q, \longrightarrow, Q_{in}, F, \Phi)$, where:

- Q is a finite and nonempty set of states.
- $\longrightarrow \subseteq Q \times Q$.
- $Q_{in} \subseteq Q$ is a set of initial states.
- $F \subseteq Q$ is a set of final states.
- $\Phi : Q \to \mathcal{M}$ is a (state-)labelling function.

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

Fig. 6. CG_M of Figure 1 (left) and $CG_{M_1 \circ M_2}$ of Figure 5 (right).

An example of an MSG is depicted in Figure 5. It is not hard to see that the language $\mathcal{L}$ defined is *not* regular. To see this, we note that $\mathcal{L}$ projected to $\{p!q, r!s\}^*$ is $\{\sigma \in \{p!q, r!s\}^* \mid |\sigma|_{p!q} = |\sigma|_{r!s} \geq 1\}$, which is not a regular string language. (Recall that regular languages are closed under projections.)

Following [2] we now define the notion of a *locally synchronized* MSG.

Communication graph For an MSC $M = (E, \leq, \lambda)$, let CG_M, *the communication graph of* M, be 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 is said to be *com-connected* if CG_M consists of one nontrivial strongly connected component and isolated vertices. An MSC language $\mathcal{L} \subseteq \mathcal{M}$ is com-connected in case each MSC $M \in \mathcal{L}$ is com-connected.

Locally synchronized MSGs The MSG $\mathcal{G}$ is *locally synchronized*[5] if for every loop $\pi = q \longrightarrow q_1 \longrightarrow \cdots \longrightarrow q_n \longrightarrow q$, the MSC $M(\pi)$ is com-connected. In our terminology, we will say that an MSC language $\mathcal{L}$ is a *locally synchronized MSG-language* if there exists a locally synchronized MSG $\mathcal{G}$ with $\mathcal{L} = \mathcal{L}(\mathcal{G})$. Figure 6 illustrates the communication graphs of the example MSCs encountered thus far. It is easy to see that neither M nor $M_1 \circ M_2$ are com-connected.

Clearly, the MSG of Figure 5 is *not* locally synchronized. This is no coincidence, as it will turn out that every locally synchronized MSG-language is a regular MSC language.

6 Finitely Generated Regular MSC Languages

As pointed in the introduction, a standard way of representing a collection of MSCS is to use an HMSC or—equivalently from a formal standpoint—as an MSG. Our goal here is to pin down the power of this representation relative to

[5] This notion is called "bounded" in [2]. The terminology "locally synchronized" is taken from [29].

the class of regular MSC languages. Stated differently we wish to characterize the class of regular MSC languages that can be represented by MSGs.

A key feature of MSG languages is that for each such language there is a fixed finite set $\mathcal{X}$ of MSCs such that each MSC in the language can be expressed as a concatenation of MSCs (with multiple copies) taken from $\mathcal{X}$. Such languages are said to be finitely generated. In this section we investigate the important connection between MSGs and finitely generated regular MSC languages. More precisely, we characterize the locally synchronized MSG-languages as precisely the class of MSC languages that are both regular and finitely generated.

Let $\mathcal{L}_1, \mathcal{L}_2 \subseteq \mathcal{M}$ be two sets of MSCs. As usual, $\mathcal{L}_1 \circ \mathcal{L}_2$ denotes the pointwise concatenation of $\mathcal{L}_1$ and $\mathcal{L}_2$, as defined out in the previous section. For $\mathcal{X} \subseteq \mathcal{M}$, we define $\mathcal{X}^0 = \{\varepsilon\}$, where ε denotes the empty MSC, and for $i \geq 0$, $\mathcal{X}^{i+1} = \mathcal{X} \circ \mathcal{X}^i$. The *asynchronous iteration of* $\mathcal{X}$ is then defined by $\mathcal{X}^{\circledast} = \bigcup_{i \geq 0} \mathcal{X}^i$. Now, let $\mathcal{L} \subseteq \mathcal{M}$. We say that $\mathcal{L}$ is *finitely generated* if there is a finite set of MSCs $\mathcal{X} \subseteq \mathcal{M}$ such that $\mathcal{L} \subseteq \mathcal{X}^{\circledast}$.

We first observe that not every regular MSC language is finitely generated. As an example, the automaton in Figure 2 accepts a regular language that is *not* finitely generated. By inspection of Figure 3 one readily verifies that none of the MSCs in this language can be expressed as the concatenation of two or more nontrivial MSCs. Hence, this language is *not* finitely generated.

Our interest in finitely generated languages stems from the fact that these arise naturally from standard high-level descriptions of MSC languages such as message sequence graphs. However, as we saw earlier, Figure 5 provides an example showing that, conversely, not all finitely generated languages are regular.

The first question we address is that of deciding whether a regular MSC language is finitely generated. To do this, we need to introduce atoms. Let $M, M' \in \mathcal{M}$ be nonempty MSCs. Then M' *is a component of* M in case there exist $M_1, M_2 \in \mathcal{M}$, possibly empty, such that $M = M_1 \circ M' \circ M_2$. Let $Comp(M)$ denote the set of components of M and let $Comp(\mathcal{L}) = \bigcup\{Comp(M) \mid M \in \mathcal{L}\}$.

We say that M *is an atom* if the only component of M is M itself. Thus, an atom is a nonempty message sequence chart that cannot be decomposed into non-trivial subcomponents. For an MSC M, we let $Atoms(M)$ denote the set $\{M' \mid M' \text{ is an atom and } M' \text{ is a component of } M\}$. For an MSC language $\mathcal{L} \subseteq \mathcal{M}$, $Atoms(\mathcal{L}) = \bigcup\{Atoms(M) \mid M \in \mathcal{L}\}$. It is clear that the question of deciding whether $\mathcal{L}$ is finitely generated is equivalent to that of checking whether $Atoms(\mathcal{L})$ is finite.

Theorem 6.1 *Let $\mathcal{L}$ be a regular MSC language. It is decidable whether $\mathcal{L}$ is finitely generated.*

Proof Sketch: Let $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$ be the minimum DFA for $\mathcal{L}$. From $\mathcal{A}$, we construct a finite family of finite-state automata that together accept the linearizations of the MSCs in $Atoms(\mathcal{L})$. It will then follow that $\mathcal{L}$ is finitely generated if and only if each of these automata accepts a finite language. We sketch the details below.

We know that for each live state $s \in S$, we can assign a capacity function $\mathcal{K}_s : Ch \rightarrow \mathbb{N}$ that counts the number of messages present in each channel when the state s is reached. We say that s is a *zero-capacity state* if $\mathcal{K}_s(c) = 0$ for each channel c.

Claim 6.2 *Let M be an MSC in $Comp(\mathcal{L})$ (in particular, in $Atoms(\mathcal{L})$) and w be a linearization of M. Then, there are zero-capacity live states s, s' in $\mathcal{A}$ such that $s \xrightarrow{w} s'$.*

If M is in $Comp(\mathcal{L})$, then there are MSC's M_1, M_2 such that $M_1 M M_2 \in \mathcal{L}$. Thus, if w_1, w_2 are some linearizations of M_1 and M_2, then $w_1 w w_2$ is accepted by $\mathcal{A}$. Thus, there is an accepting run $s_{in} \xrightarrow{w_1} s \xrightarrow{w} s' \xrightarrow{w_2} t$. w_1, w_2 and w are complete words as they arise as linearizations of MSCs. Further, s_{in} is a zero-capacity state and thus s and s' must be zero-capacity states. This proves Claim 6.2.

Claim 6.3 *Let M be an MSC in $Comp(\mathcal{L})$. M is an atom if and only if for each linearization w of M and each pair (s, s') of zero-capacity live states in $\mathcal{A}$, if $s \xrightarrow{w} s'$ then no intermediate state visited in this run has zero-capacity.*

Let M an atom and w be a linearization of M. Suppose $w = w_1 w_2$ for nonempty words w_1 and w_2 and $s \xrightarrow{w_1} s_1 \xrightarrow{w_2} s'$, where s_1 is a zero-capacity state. w_1 and w_2 are nonempty complete words. Recall that every complete word is the linearization of some MSC. Let M_1 and M_2 be the MSCs corresponding to w_1 and w_2. Then, $M = M_1 \circ M_2 \circ M_3$, where M_3 is the empty MSC, contradicting the assumption that M is an atom. Thus, the run can have no intermediate zero-capacity state.

Suppose M is not an atom. Then $M = M_1 \circ M_2 \circ M_3$ where at least two of M_1, M_2, M_3 are nonempty. Let w_1, w_2 and w_3 be linearizations of M_1, M_2 and M_3. All three are complete words. Thus, there are states s_1, s_2 such that $s \xrightarrow{w_1} s_1 \xrightarrow{w_2} s_2 \xrightarrow{w_3} s'$. Since at least one of these words is nonempty, one of the states s_1 or s_2 is a zero-capacity intermediate state. This completes the proof of Claim 6.3.

Suppose $s \xrightarrow{w} s'$ and $w \sim w'$. Then it is easy to see that $s \xrightarrow{w'} s'$ as well. With each pair (s, s') of live zero-capacity states we associate a language $L_{At}(s, s')$. A word w belongs to $L_{At}(s, s')$ if and only if w is complete, $s \xrightarrow{w} s'$ and for each $w' \sim w$ the run $s \xrightarrow{w'} s'$ has no zero-capacity intermediate states. From Claims 6.2 and 6.3 above, each of these languages consists of all the linearizations of some subset of $Atoms(\mathcal{L})$ and the linearizations of each element of $Atoms(\mathcal{L})$ is contained in some $L_{At}(s, s')$. Thus, it suffices to check for the finiteness of each of these languages.

Let $L_{s,s'}$ be the language of strings accepted by $\mathcal{A}$ when we set the initial state to be s and the set of final states to be $\{s'\}$. Clearly $L_{At}(s, s') \subseteq L_{s,s'}$. We now show how to construct an automaton for $L_{At}(s, s')$.

We begin with $\mathcal{A}$ and prune the automaton as follows:

- Remove all incoming edges at s and all outgoing edges at s'.
- If $t \notin \{s, s'\}$ and $\mathcal{K}_t = \overline{0}$, remove t and all its incoming and outgoing edges.
- Recursively remove all states that become unreachable as a result of the preceding operation.

Let $\mathcal{B}$ be the resulting automaton. $\mathcal{B}$ accepts any complete word w on which the run from s to s' does not visit an intermediate zero-capacity state. Clearly, $L_{At}(s, s') \subseteq L(\mathcal{B})$. However, $L(\mathcal{B})$ may also contain linearizations of non-atomic MSCs that happen to have no nontrivial complete prefix. For all such words, we know from Claim 6.3 that there is at least one equivalent linearization on which the run passes through a zero-capacity state and which would hence be eliminated from $L(\mathcal{B})$. Thus, $L_{At}(s, s')$ is the $\sim$-closed subset of $L(\mathcal{B})$ and we need to prune $\mathcal{B}$ further to obtain the automaton for $L_{At}(s, s')$.

Recall that the original DFA $\mathcal{A}$ was structurally closed with respect to the independence relation on communication actions in the following sense. Suppose $\delta(s_1, a) = s_2$ and $\delta(s_2, b) = s_3$ with a, b independent at s_1. Then, there exists s_2' such that $\delta(s_1, b) = s_2'$ and $\delta(s_2', a) = s_3$.

To identify the closed subset of $L(\mathcal{B})$, we look for local violations of this "diamond" property and carefully prune transitions. We first blow up the state space into triples of the form (s_1, s_2, s_3) for each s_1, s_2 and s_3 such that there exist a and a' with $\delta(s_1, a) = s_2$ and $\delta(s_2, a') = s_3$. Let S' denote this set of triples. We obtain a nondeterministic transition relation $\delta' = \{((s_1, s_2, s_3), a, (t_1, t_2, t_3)) \mid s_2 = t_1, s_3 = t_2, \delta(s_2, a) = s_3\}$. Set $S_{in} = \{(s_1, s_2, s_3) \in S' \mid s_2 = s_{in}\}$ and $F' = \{(s_1, s_f, s_2) \in S' \mid s_f \in F\}$. Let $\mathcal{B}' = (S', \Sigma, \delta', S_{in}, F')$.

Consider any state s_1 in $\mathcal{B}$ such that a and b are independent at s_1, $\delta(s_1, a) = s_2$, $\delta(s_2, b) = s_3$ but there is no s_2' such that $\delta(s_1, b) = s_2'$ and $\delta(s_2', a) = s_3$. For each such s_1, we remove all transitions of the form $((t, s_0, s_1), a, (s_0, s_1, t'))$ and $((t, s_2, s_3), b, (s_2, s_3, t'))$ from $\mathcal{B}'$. We then recursively remove all states that become unreachable after this pruning.

Eventually, we arrive at an automaton $\mathcal{C}$ such that $L(\mathcal{C}) = L_{At}(s, s')$. Since $\mathcal{C}$ is a finite-state automaton, we can easily check whether $L(\mathcal{C})$ is finite. This process is repeated for each pair of live zero-capacity states. $\Box$

We will now bring out the intimate connection between message sequence graphs and finitely generated regular MSC languages. As pointed out earlier, Alur and Yannakakis noted that every locally synchronized MSG-language is regular [2, Thm. 7]. One way to establish this result is — following [9] — to show that the asynchronous iteration of a com-connected regular MSC language is regular. The proof in [9] is based on grammars. A more direct, automata-theoretic proof of the same result is described in Appendix A. Thus, every locally synchronized MSG accepts a finitely generated regular MSC language.

All languages arising from MSGs are finitely generated, so the language accepted by the message-passing automaton on Figure 2 shows that not all regular MSC languages can be described by MSGs. It turns out that locally synchronized MSGs generate precisely those MSC languages that are both regular and finitely generated.

Theorem 6.4 *Let $\mathcal{L}$ be an MSC language. Then $\mathcal{L}$ is a finitely generated regular MSC language if and only if $\mathcal{L}$ is a locally synchronized MSG-language.*

Proof Sketch: From the remarks above, it suffices to show that any finitely generated regular MSC language can be accepted by some locally synchronized MSG.

Suppose $\mathcal{L}$ is a regular MSC language accepted by the minimal DFA $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$. Let $Atoms(\mathcal{L}) = \{a_1, a_2, \ldots, a_m\}$. For each atom a_i, fix a linearization $u_i \in lin(a_i)$. Define an auxiliary DFA $\mathcal{B} = (S^{\overline{0}}, Atoms(\mathcal{L}), s_{in}, \widehat{\delta}, \widehat{F})$ as follows:

- $S^{\overline{0}}$ is the set of states of $\mathcal{A}$ that have zero-capacity functions.
- $\widehat{F} = F$.
- $\widehat{\delta}(s, a_i) = s'$ iff $\delta(s, u_i) = s'$ in $\mathcal{A}$. (Note that $u, u' \in lin(a_i)$ implies $\delta(s, u) = \delta(s, u')$, so s' is fixed independent of the choice of $u_i \in lin(a_i)$.)

Thus, $\mathcal{B}$ accepts the (regular) language of atoms corresponding to $\mathcal{L}(\mathcal{A})$. We can define a natural independence relation I_A on atoms as follows: atoms a_i and a_j are independent if and only if the set of active processes in a_i is disjoint from the set of active processes in a_j. (The process p is *active* in the MSC $(E, \leq, \lambda)$ if E_p is non-empty.)

It follows that $L(\mathcal{B})$ is a regular Mazurkiewicz trace language over the trace alphabet $(Atoms(\mathcal{L}), I_A)$. As usual, for $w \in Atoms(\mathcal{L})^*$, we let $[w]$ denote the equivalence class of w with respect to I_A.

We now fix a strict linear order $\prec$ on $Atoms(\mathcal{L})$. This induces a (lexicographic) total order on words over $Atoms(\mathcal{L})$. Let $LEX \subseteq Atoms(\mathcal{L})^*$ be given by: $w \in LEX$ iff w is the lexicographically least element in $[w]$.

For a trace language L over $(Atoms(\mathcal{L}), I_A)$, let $lex(L)$ denote the set $L \cap LEX$.

Fact 6.5 ([11], Sect. 6.3.1)

(1) If L is a regular trace language over $(Atoms(\mathcal{L}), I_A)$, then $lex(L)$ is a regular language over $Atoms(\mathcal{L})$. Moreover, $L = \{[w] \mid w \in lex(L)\}$.
(2) If $w_1 w w_2 \in LEX$, then $w \in LEX$.
(3) If w is not *a connected*[6] *trace, then $ww \notin LEX$.*

From (1) we know that $lex(L(\mathcal{B}))$ is a regular language over $Atoms(\mathcal{L})$. Let $\mathcal{C} = (S', Atoms(\mathcal{L}), s'_{in}, \delta', F')$ be the DFA over $Atoms(\mathcal{L})$ obtained by eliminating the (unique) dead state, if any, from the minimal DFA for $lex(L(\mathcal{B}))$. It is easy to see that an MSC M belongs to $\mathcal{L}$ if and only if it can be decomposed into a sequence of atoms accepted by $\mathcal{C}$. Using this fact, we can derive an MSG $\mathcal{G}$ from $\mathcal{C}$ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}$. We define $\mathcal{G} = (Q, \longrightarrow, Q_{in}, F, \Phi)$ as follows:

- $Q = S' \times (Atoms(\mathcal{L}) \cup \{\varepsilon\})$.
- $Q_{in} = \{(s'_{in}, \varepsilon)\}$.
- $(s, b) \longrightarrow (s', b')$ iff $\delta'(s, b') = s'$.
- $F' = F \times Atoms(\mathcal{L})$.
- $\Phi(s, b) = b$.

Clearly $\mathcal{G}$ is an MSG and the MSC language that it defines is $\mathcal{L}$. We need to show that $\mathcal{G}$ is locally synchronized. To this end, let $\pi = (s, b) \longrightarrow (s_1, b_1) \longrightarrow \cdots \longrightarrow (s_n, b_n) \longrightarrow (s, b)$ be a loop in $\mathcal{G}$. We need to establish that the MSC $M(\pi) = b_1 \circ \cdots \circ b_n \circ b$ defined by this loop is com-connected. Let $w = b_1 b_2 \ldots b_n b$.

Consider the corresponding loop $s \xrightarrow{b_1} s_1 \xrightarrow{b_2} \cdots \xrightarrow{b_n} s_n \xrightarrow{b} s$ in $\mathcal{C}$. Since every state in $\mathcal{C}$ is live, there must be words w_1, w_2 over $Atoms(\mathcal{L})$ such that $w_1 w^k w_2 \in lex(L(\mathcal{B}))$ for every $k \geq 0$.

From (2) of Fact 6.5, $w^k \in LEX$. This means, by (3) of Fact 6.5, that

[6] A trace is said to be *connected* if, when viewed as a labeled partial order, its Hasse diagram consists of a single connected component. See [11] for a more formal definition.

w describes a connected trace over $(Atoms(\mathcal{L}), I_A)$. Further, the underlying undirected graph of the communication graph of any atom always consists of a single nontrivial connected component. From these, it is not difficult to see that the underlying undirected graph of the communication graph $CG_{M(\pi)} = (\mathcal{P}, \mapsto)$ consists of a single connected component $C \subseteq \mathcal{P}$ and isolated processes. We have to argue that the component C is, in fact, *strongly* connected. We show that if C is not strongly connected, then the regular MSC language $\mathcal{L}$ is not B-bounded for any $B \in \mathbb{N}$, thus contradicting Proposition 2.4.

Suppose that the underlying graph of C is connected but C not strongly connected. Then, there exist two processes $p, q \in C$ such that $p \mapsto q$, but there is no path from q back to p in $CG_{M(\pi)}$. For $k \geq 0$, let $M(\pi)^k = (E, \leq, \lambda)$ be the MSC corresponding to the k-fold iteration $\underbrace{M(\pi) \circ M(\pi) \circ \cdots \circ M(\pi)}_{k \text{ times}}$.
Since $p \mapsto q$ in $CG_{M(\pi)}$, it follows that there are events labeled $p!q$ and $q?p$ in $M(\pi)$. Moreover, since there is no path from q back to p in $CG_{M(\pi)}$, we can conclude that in $M(\pi)^k$, for each event e with $\lambda(e) = p!q$, there is no event labeled $q?p$ in $\downarrow e$. This means that $M(\pi)^k$ admits a linearization v'_k with a prefix τ'_k that includes all the events labeled $p!q$ and excludes all the events labeled $q?p$, so that $\#_{p!q}(\tau) - \#_{q?p}(\tau) \geq k$.

By Proposition 2.4, since $\mathcal{L}$ is a regular MSC language, there is a bound $B \in \mathbb{N}$ such that every word in $\mathcal{L}$ is B-bounded—that is, for each $v \in \mathcal{L}$, for each prefix τ of v and for each channel $(p, q) \in Ch$, $\#_{p!q}(\tau) - \#_{q?p}(\tau) \leq B$. Recall that $w_1 w^k w_2 \in lex(L(\mathcal{B}))$ for every $k \geq 0$. Fix linearizations v_1 and v_2 of the atom sequences w_1 and w_2, respectively. Then, for every $k \geq 0$, $u_k = v_1 v'_k v_2 \in \mathcal{L}$ where v'_k is the linearization of $M(\pi)^k$ defined earlier. Setting k to be $B{+}1$, we find that u_k admits a prefix $\tau_k = v_1 \tau'_k$ such that $\#_{p!q}(\tau_k) - \#_{q?p}(\tau_k) \geq B{+}1$, which contradicts the B-boundedness of $\mathcal{L}$.

Hence, it must be the case that C is a strongly connected component, which establishes that the MSG $\mathcal{G}$ we have constructed is locally synchronized.

□

It is easy to see that local synchronicity is not a necessary condition for regularity. Consider the MSG in Figure 7, which is not locally synchronized. It accepts the regular MSC language $M_1 \circ (M_1 + M_2)^{\circledast}$.

Thus, it would be useful to provide a characterization of the class of MSGs representing regular MSC languages. Unfortunately, the following result shows that there is no (recursive) characterization of this class.

Theorem 6.6 *The problem of deciding whether a given MSG represents a*

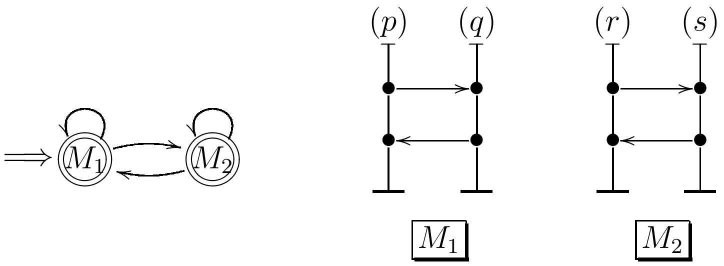

Fig. 7. An non-locally synchronized MSG whose language is regular.

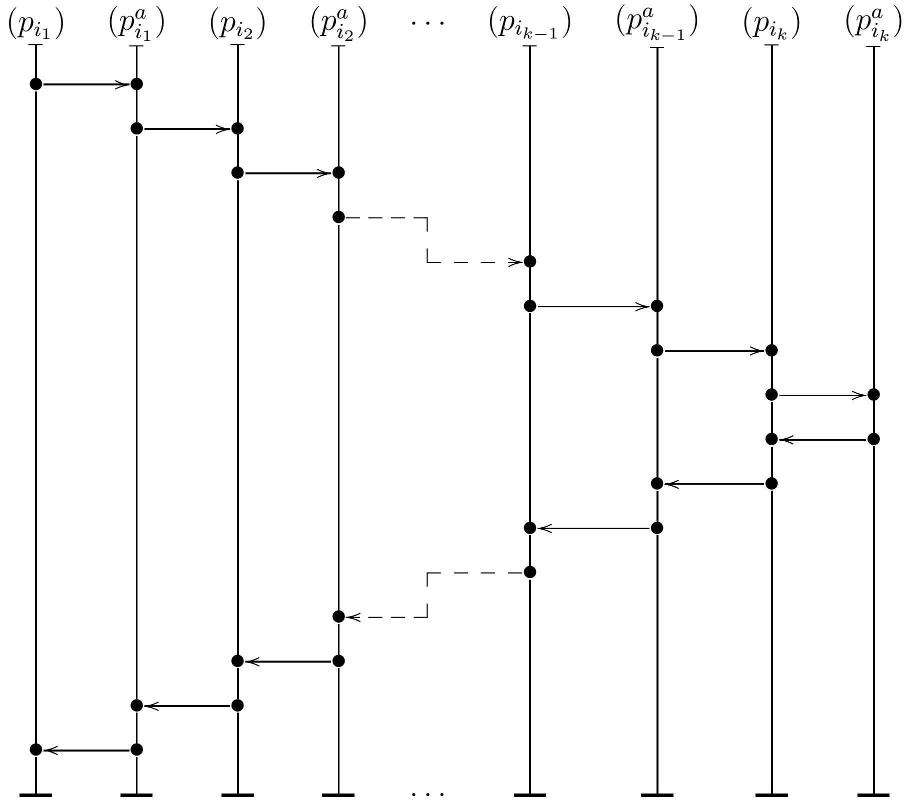

Fig. 8. The MSC M_a encoding the letter $a \in A$.

regular MSC language is undecidable.

Proof Sketch: It is known that the problem of determining whether the trace-closure of a regular language $L \subseteq A^*$ with respect to a trace alphabet (A, I) is also regular is undecidable [32]. We reduce this problem to the problem of checking whether the MSC language defined by an MSG is regular.

Let $\widetilde{A} = (A_1, \ldots, A_n)$ be a distributed alphabet implementing the trace alphabet (A, I) [11]. We will fix a set of processes $\mathcal{P}$ and the associated communication alphabet Σ and encode each letter a by an MSC M_a over $\mathcal{P}$.

For each component A_i of $\widetilde{A}$, we create $1+|A_i|$ processes that we will denote by $p_i, p_i^{a_1}, p_i^{a_2}, \ldots, p_i^{a_k}$, where $A_i = \{a_1, a_2, \ldots, a_k\}$. Suppose now that the letter a appears in the components $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ of the distributed alphabet $\widetilde{A}$ with $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. The MSC M_a representing a is then given in Figure 8. It is easy to see that the communication graph CG_{M_a} is strongly connected. Moreover, if $(a, b) \in I$, then the sets of active processes of M_a and M_b are disjoint. The encoding ensures that we can construct a finite-state automaton to parse any word over Σ and determine whether it arises as the linearization of an MSC of the form $M_{a_1} \circ M_{a_2} \circ \cdots \circ M_{a_k}$. If so, the parser can uniquely reconstruct the corresponding word $a_1 a_2 \ldots a_k$ over A.

Let $\mathcal{A}$ be the minimal DFA corresponding to a regular language L over A. We construct an MSG $\mathcal{G}$ from $\mathcal{A}$ as described in the proof of Theorem 6.4. Given the properties of our encoding, we can then establish that the MSC language $L(\mathcal{G})$ is regular if and only if the trace-closure of L is regular, thus completing the reduction. $\square$

7 Conclusion

We have identified here the notion of a regular MSC language and have developed the basic theory of these languages by providing automata-theoretic and logical chracterizations. We have also characterized precisely the subclass of regular MSC languages definable using the mechanism of HMSCs. Our range of results shows that the notion of regularity that we have identified here is a fruitful one. Further, while it bears a pleasant similarity to the theory of regular Mazurkiewicz trace languages, its theory requires new insights and techniques due to the implicit presence of potentially unbounded FIFOs.

Our treatment of MSC languages and the related work cited so far have implicitly assumed a linear time framework. The notion of an implementation (say an MPA) satisfying a requirement specification (say, a bounded HMSC) is also an *existential* one; for every MSC in the requirement there exists an MSC in the implementation and conversely. The formalism of Live Sequence Charts proposed by Damm and Harel [10] suggests, however, that one could obtain a more powerful specification language based on MSCs by switching to a branching-time framework. The recent work of Harel and his collaborators [15,16] suggests that this way of using MSCs might bear a more direct and fruitful relationship with implementations than mechanisms such as HMSCs or sequence diagrams in the UML framework. In light of this, it will be interesting to formulate a suitable branching-time version of the theory reported in this paper.

References

[1] Alur, R., Holzmann, G. J., Peled, D.: An analyzer for message sequence charts. *Software Concepts and Tools* **17**(2) (1996) 70–77

[2] Alur, R., Yannakakis, M.: Model checking of message sequence charts. *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, Lecture Notes in Computer Science **1664**, Springer-Verlag (1999) 114–129

[3] Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence graphs. *Proceedings of the 22nd International Conference on on Software Engineering (ICSE 2000)*, Association for Computing Machinery (2000) 304–313.

[4] Alur, R., Etessami, K., Yannakakis, M.: Realizability and Verification of MSC Graphs. *Proceedings Automata, Languages and Programming, 28th International Colloquium (ICALP 2001)*, Lecture Notes in Computer Science **2076**, Springer-Verlag (2001) 797–808.

[5] Ben-Abdallah, H., Leue, S.: Syntactic detection of process divergence and non-local choice in message sequence charts. *Proceedings of the 3rd Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, Lecture Notes in Computer Science **1217**, Springer-Verlag (1997) 259–274

[6] Bollig, B., Leucker, M., Noll, T.: Generalised regular MSC languages. *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'02)*, Lecture Notes in Computer Science **2303**, Springer-Verlag (2002) 52–66

[7] Booch, G., Jacobson, I., Rumbaugh, J.: *Unified Modeling Language User Guide.* Addison-Wesley (1997)

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

[9] Clerbout, M., Latteux, M.: Semi-commutations. *Information and Computation* **73**(1) (1987) 59–74

[10] Damm, W., Harel, D.: LSCs: Breathing life into message sequence charts. *Formal Methods in System Design* **19(1)** (2001) 45–80.

[11] Diekert, V., Rozenberg, G. (Eds.): *The Book of Traces.* World Scientific (1995)

[12] Ebinger, W., Muscholl, A.: Logical definability on infinite traces. *Theoretical Computer Science* **154**(1) (1996) 67–84

[13] Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-state high-level MSCs: Model-checking and realizability. *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)*, Lecture Notes in Computer Science **2380**, Springer-Verlag (2002) 657–668

[14] Harel, D., Gery, E.: Executable object modeling with statecharts. *IEEE Computer*, July 1997 (1997) 31–42

[15] Harel, D., Marelly, R.: Specifying and executing behavioral requirements: The play-in/play-out approach. *Software and System Modeling (SoSyM)* (to appear)

[16] Harel, D., Kugler, H., Marelly, R., Pnueli, A.: Smart Play-out of Behavioral Requirements. *Proceedings Formal Methods in Computer-Aided Design, 4th International Conference (FMCAD 2002)*, Lecture Notes in Computer Science **2517** Springer-Verlag (2002) 378–398

[17] Henriksen, J.G., Mukund, M., Narayan Kumar, K., Thiagarajan, P.S.: On message sequence graphs and finitely generated regular MSC languages. *Proceedings of the International Colloquium on Automata, Languages and Programming 2000 (ICALP'00)*, Lecture Notes in Computer Science **1854**, Springer-Verlag (2000) 675–686

[18] Henriksen, J.G., Mukund, M., Narayan Kumar, K., Thiagarajan, P.S.: Regular collections of message sequence charts. *Proceedings of the Conference on the Mathematical Foundations of Computer Science 2000 (MFCS'00)*, Lecture Notes in Computer Science **1893**, Springer-Verlag (2000) 405–414

[19] ITU-TS Recommendation Z.120: *Message Sequence Chart (MSC)*. ITU-TS, Geneva (1997)

[20] Kuske, D.: A further step towards a theory of regular MSC languages. *Proceedings of the Symposium on the Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science **2285**, Springer-Verlag (2002) 489–500

[21] Ladkin, P. B., Leue, S.: Interpreting message flow graphs. *Formal Aspects of Computing* **7**(5) (1995) 473–509

[22] Levin, V., Peled, D.: Verification of message sequence charts via template matching. *Proceedings of the 7th International Conference on Theory and Practice of Software Development (TAPSOFT'97)*, Lecture Notes in Computer Science **1214**, Springer-Verlag (1997) 652–666

[23] Madhusudan, P.: Reasoning about sequential and branching behaviours of message sequence graphs. *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, Lecture Notes in Computer Science **2076**, Springer-Verlag (2001) 396–407

[24] Madhusudan, P., Meenakshi, B.: Beyond message sequence graphs. *Proceedings of the 21st Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01)*, Lecture Notes in Computer Science **2245**, Springer-Verlag (2001) 256–267

[25] Mauw, S., Reniers, M. A.: High-level message sequence charts, *Proceedings of the 8th SDL Forum, SDL'97: Time for Testing — SDL, MSC and Trends*, Elsevier (1997) 291–306

[26] Mukund, M., Narayan Kumar, K., Sohoni, M.: Synthesizing distributed finite-state systems from MSCs. *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000)*, Lecture Notes in Computer Science **1877**, Springer-Verlag (2000) 521–535

[27] Mukund, M., Narayan Kumar, K., Sohoni, M.: Bounded time-stamping in message-passing systems. *Theoretical Computer Science*, **290(1)** (2003) 221–239

[28] Muscholl, A.: Matching specifications for message sequence charts. *Proceedings of the 2nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'99)*, Lecture Notes in Computer Science **1578**, Springer-Verlag (1999) 273–287

[29] Muscholl, A., Peled, D.: Message sequence graphs and decision problems on Mazurkiewicz traces. *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science (MFCS'99)*, Lecture Notes in Computer Science **1672**, Springer-Verlag (1999) 81–91

[30] Muscholl, A., Peled, D., Su, Z.: Deciding properties for message sequence charts. *Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structures (FOSSACS'98)*, Lecture Notes in Computer Science **1378**, Springer-Verlag (1998) 226–242

[31] Rudolph, E., Graubmann, P., Grabowski, J.: Tutorial on message sequence charts. In *Computer Networks and ISDN Systems — SDL and MSC* **28** (1996).

[32] Sakarovitch, J.: The "last" decision problem for rational trace languages. *Proceedings of the 1st Latin American Symposium on Theoretical Informatics (LATIN'92)*, Lecture Notes in Computer Science **583**, Springer-Verlag (1992) 460–473

[33] Thiagarajan, P. S.: A trace consistent subset of PTL. *Proceedings of 6th International Conference on Concurrency Theory (CONCUR'95)*, Lecture Notes in Computer Science **962**, Springer-Verlag (1995) 438-452

[34] Thiagarajan, P. S., Walukiewicz, I.: An expressively complete linear time temporal logic for Mazurkiewicz traces. *Information and Computation*, Vol. 179, No. 2 (2002) 230-249.

[35] Thomas, W.: Automata on infinite objects. In van Leeuwen, J. (Ed.): *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, Elsevier Science Publishers (1990) 133–191

[36] Thomas, W.: Languages, automata, and logic. In Rozenberg, G., Salomaa, A. (Eds.): *Handbook of Formal Language Theory, Volume III*, Springer-Verlag (1997) 389–455

[37] Vardi, M. Y., Wolper, P.: An automata-theoretic approach to automatic program verification. *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science (LICS'86)*, IEEE Computer Society Press (1986) 332–345

[38] Zielonka, W.: Notes on finite asynchronous automata. *R.A.I.R.O. Informatique Théorique et Applications* **21** (1987) 99–135

A Asynchronous Iteration

In this section, we give an automata-theoretic proof that the asynchronous iteration of a com-connected regular MSC language remains regular. A proof of this result in terms of grammars appears in [9].

We begin with a simple characterization of asynchronous iteration that follows from the definition in Section 6.

Proposition A.1 *Let $\mathcal{L} \subseteq \mathcal{M}$ be an MSC language. The MSC $M = (E, \leq, \lambda)$ belongs to $\mathcal{L}^{\circledast}$, the* asynchronous iteration *of $\mathcal{L}$, iff there is a sequence of complete ideals $\emptyset = I_0 \subset I_1 \subset \cdots \subset I_n = E$ such that for each $j \in \{1, 2, \ldots, n\}$, the partial order $I_j \setminus I_{j-1}$ is isomorphic to some $M' \in \mathcal{L}$.*

The ideals $I_0 I_1 \ldots I_n$ define an *$\mathcal{L}$-factorization* of M—that is, a factorization of M into MSCs from $\mathcal{L}$.

A.1 An infinite-state automaton for $\mathcal{L}^{\circledast}$

Let $\mathcal{L}$ be a regular MSC language. From the automata-theoretic characterization of Section 3, it follows that there is a B-bounded message-passing automaton $\mathcal{A}$ such that $L(\mathcal{A}) = \mathcal{L}$. To construct a (sequential) automaton for $\mathcal{L}^{\circledast}$, our strategy will be to guess a factorization of the input and simulate $\mathcal{A}$ to verify that each factor belongs to $\mathcal{L}$. We first construct an infinite-state automaton for $\mathcal{L}^{\circledast}$ for an arbitrary regular MSC language $\mathcal{L}$ and then describe the conditions under which we can restrict the automaton for $\mathcal{L}^{\circledast}$ to be a finite-state device.

The new automaton $\mathcal{A}^{\circledast}$ that we construct uses natural numbers to label the factors. Since not every process participates in every factor, $\mathcal{A}^{\circledast}$ records the sequence of factors that each process $p \in \mathcal{P}$ participates in and ensures that the sequence in which the factors are processed is consistent across the system. In addition, $\mathcal{A}^{\circledast}$ simulates a copy of $\mathcal{A}$ on each factor. Initially, each factor is labelled by the initial configuration of $\mathcal{A}$. The simulation succeeds if the global state associated with each factor is a final configuration of $\mathcal{A}$.

More formally, $\mathcal{A}^{\circledast} = (S', s'_{in}, \longrightarrow', F')$ where each state in S' is a pair (μ, ν) with $\mu : \mathcal{P} \to \mathbb{N}^*$ and $\nu : \mathbb{N} \to \mathit{Conf}_{\mathcal{A}}$ such that μ satisfies the following condition:

- For any pair of processes p and q (not necessarily distinct) and any pair of distinct labels ℓ and ℓ', if ℓ appears before ℓ' in $\mu(p)$, then ℓ' does not appear before ℓ in $\mu(q)$.

The function μ records the order in which each process observes the $\mathcal{L}$-factors of the input word. The function ν keeps track of the current configuration of $\mathcal{A}$ on each factor.

The initial state s'_{in} of $\mathcal{A}^{\circledast}$ is the pair (μ_{in}, ν_{in}) where $\mu_{in}(p) = \varepsilon$ for each process p and $\nu_{in}(\ell) = (s_{in}, \chi_\varepsilon)$ for each $\ell \in \mathbb{N}$ (where ε is the empty word and $(s_{in}, \chi_\varepsilon)$ is the initial configuration of $\mathcal{A}$).

A state (μ, ν) of $\mathcal{A}^{\circledast}$ is in F' whenever:

- If ℓ appears in $\mu(p)$ for some process p, $\nu(\ell)$ is a final configuration of $\mathcal{A}$.
- If ℓ does not appear in $\mu(p)$ for any process p, $\nu(\ell) = (s_{in}, \chi_\varepsilon)$.

Consider states (μ, ν) and (μ', ν') and a letter a such that $a \in \Sigma_p$. Then, $(\mu, \nu) \xrightarrow{a}' (\mu', \nu')$ provided:

- For $q \neq p$, $\mu'(q) = \mu(q)$.
- Either $\mu'(p) = \mu(p)$ or $\mu'(p) = \mu(p) \cdot \ell$ for some $\ell \in \mathbb{N}$.
- Let the last label in $\mu'(p)$ be ℓ. Then, $\nu(\ell) \overset{a}{\Longrightarrow} \nu'(\ell)$ and for $\ell' \neq \ell$, $\nu'(\ell') = \nu(\ell')$ (where $\Longrightarrow \subseteq \mathit{Conf}_{\mathcal{A}} \times \Sigma \times \mathit{Conf}_{\mathcal{A}}$ is the global transition relation of $\mathcal{A}$).

The following is easy to verify from the definition of $\mathcal{A}^{\circledast}$.

Theorem A.2 *Let $\mathcal{A}$ be a message-passing automaton for a regular MSC language $\mathcal{L}$. Then, the automaton $\mathcal{A}^{\circledast}$ accepts the language $\mathcal{L}^{\circledast}$.*

To describe when we can restrict $\mathcal{A}^{\circledast}$ to a finite-state device, we extend the definition of $\mathcal{A}^{\circledast}$ so that each state has one more component. A state of $\mathcal{A}^{\circledast}$ is now a triple of functions (μ, ν, τ), where $\mu : \mathcal{P} \to \mathbb{N}^*$ and $\nu : \mathbb{N} \to \mathit{Conf}_{\mathcal{A}}$ are as before. The new component $\tau : \mathbb{N} \to 2^{\mathcal{P}}$ specifies the *type* of each label.

As before, μ records the sequence in which each process observes $\mathcal{L}$-factors while ν keeps track of the current configuration of each factor. The new component τ records the set of processes that participate in each factor.

The states of $\mathcal{A}^{\circledast}$ are those triples (μ, ν, τ) that satisfy the following conditions:

- For any pair of processes p and q (not necessarily distinct) and any pair of distinct labels ℓ and ℓ', if ℓ appears before ℓ' in $\mu(p)$, then ℓ' does not appear before ℓ in $\mu(q)$.
- If $\tau(\ell) \neq \emptyset$ then ℓ appears in $\mu(p)$ for some $p \in \mathcal{P}$. Moreover, if ℓ appears in $\mu(p)$ then $p \in \tau(\ell)$.

The initial state s'_{in} of the extended version of $\mathcal{A}^{\circledast}$ is the triple $(\mu_{in}, \nu_{in}, \tau_{in})$ where $\mu_{in}(p) = \varepsilon$ for each process p, $\nu_{in}(\ell) = (s_{in}, \chi_\varepsilon)$ for each $\ell \in \mathbb{N}$ and $\tau_{in}(\ell) = \emptyset$ for each $\ell \in \mathbb{N}$.

A state (μ, ν, τ) of $\mathcal{A}^{\circledast}$ is in F' whenever:

- If ℓ appears in $\mu(p)$ for some process p, $\nu(\ell)$ is a final configuration of $\mathcal{A}$.
- If ℓ does not appear in $\mu(p)$ for any process p, $\nu(\ell) = (s_{in}, \chi_{\varepsilon})$.
- If $\tau(\ell) = P$ then ℓ appears in $\mu(p)$ for each $p \in P$.

Consider states (μ, ν, τ) and (μ', ν', τ') and a letter a such that $a \in \Sigma_p$. Then, $(\mu, \nu, \tau) \xrightarrow{a}' (\mu', \nu', \tau')$ provided:

- For $q \neq p$, $\mu'(q) = \mu(q)$.
- Either $\mu'(p) = \mu(p)$ or $\mu'(p) = \mu(p) \cdot \ell$ for some $\ell \in \mathbb{N}$.
- Let the last label in $\mu'(p)$ be ℓ. Then, $\nu(\ell) \stackrel{a}{\Longrightarrow} \nu'(\ell)$ and for all $\ell' \neq \ell$, $\nu'(\ell) = \nu(\ell)$.
- Let the last label in $\mu'(p)$ be ℓ. Then $\tau'(\ell) \supset \{p\}$ and for $\ell' \neq \ell$, $\tau'(\ell') = \tau(\ell')$. Moreover, if ℓ already appears in $\mu(q)$ for some $q \in \mathcal{P}$, then $\tau'(\ell) = \tau(\ell)$. (This captures the fact that when ℓ is first used, $\tau(\ell)$ records a nondeterministic guess for the processes that will participate in the factor labelled ℓ and this guess cannot be changed.)

Once again, we can establish that $L(\mathcal{A}^{\circledast}) = L(\mathcal{A})^{\circledast}$.

A.2 If $\mathcal{L}$ is com-connected, $\mathcal{L}^{\circledast}$ is regular

Recall the definition of a com-connected MSC language from Section 5. The main result we want to prove is the following.

Theorem A.3 *Let $\mathcal{L}$ be a regular and com-connected MSC language. Then, $\mathcal{L}^{\circledast}$ is regular.*

In the previous section, we saw how to construct an infinite-state automaton $\mathcal{A}^{\circledast}$ for $\mathcal{L}^{\circledast}$ from a message-passing automaton $\mathcal{A}$ for $\mathcal{L}$. To prove Theorem A.3, we shall argue that if $\mathcal{L}$ is com-connected, $\mathcal{A}^{\circledast}$ can in fact be cut down to a finite-state automaton.

Definition A.4 Let $G = (V, E)$ be a directed graph. For $X \subseteq V$, define $nbd(X)$, the *neighbourhood* of X, to be $X \cup \{v' \mid \exists v \in X : (v', v) \in E\}$. $\Box$

Proposition A.5 *Let $G = (V, E)$ be a directed graph such that all non-isolated vertices form a single strongly connected component. Let $C \subseteq V$ be the vertices in this strongly connected component. Then, for any proper subset $C' \subsetneq C$, $nbd(C')$ has at least one vertex in $C \setminus C'$.*

Proof: Suppose that $C' \subsetneq C$ but there is no vertex $v \in (C \setminus C') \cap nbd(C')$. This means there is no path from any vertex in $C \setminus C'$ to any vertex in C'. This contradicts the assumption that C is a strongly connected component of G. □ □

Definition A.6 Consider a state (μ, ν, τ) of the extended automaton $\mathcal{A}^{\circledast}$ described in the previous section. The label ℓ is said to be *dead* in (μ, ν, τ) if for every $p \in \tau(\ell)$, $\mu(p) = w \cdot \ell \cdot w'$, where w' is a nonempty string over $\mathbb{N}$. A label that is not dead is said to be *live*. □

Lemma A.7 *Let $\mathcal{A}$ be a message-passing automaton for a com-connected MSC language $\mathcal{L}$. In any state (μ, ν, τ) of $\mathcal{A}^{\circledast}$ only a bounded number of labels are not dead.*

Proof: Let (μ, ν, τ) be a state of $\mathcal{A}^{\circledast}$ and let $p \in \mathcal{P}$. Suppose that $\mu(p)$ is of the form $u_0\ell_0u_1\ell_1 \ldots \ell_k u_k \ell_{k+1} u_{k+1}$, where each u_i, $i \in \{0, 1, \ldots, k{+}1\}$, is a string over $\mathbb{N}$, $\tau(\ell_0) = \tau(\ell_1) = \cdots = \tau(\ell_{k+1}) = P$ and $|P| = k$. Then, ℓ_0 must be dead.

Recall that for each ℓ, $\tau(\ell)$ records the set of processes that participate in the factor M_ℓ labelled ℓ. Since $\mathcal{L}$ is com-connected, $\tau(\ell)$ defines a strongly connected set of processes in $CG(M_\ell)$.

Consider the graph $G_{M_{\ell_k}}$. Let $P_k = nbd(p)$ in this graph. For each process $q \in P_k$, there is an edge from q to p in $G_{M_{\ell_k}}$. Thus, there is at least one action $p?q$ in the factor M_{ℓ_k}. Since p has progressed from the factor M_{ℓ_k} to the factor $M_{\ell_{k+1}}$, the corresponding q-action $q!p$ in M_{ℓ_k} must also have occurred already. Thus, q has also observed the factor ℓ_k and ℓ_k must appear in $\mu(q)$ as well.

Let $P_{k-1} = nbd(P_k)$ in $G_{M_{\ell_k}}$. By a similar argument, ℓ_{k-1} must appear in $\mu(q)$ for each $q \in P_{k-1}$.

In this vein, we can construct $P_{k-2}, P_{k-3}, \ldots$ such that for each $j \in \{k, k{-}1, \ldots, 1\}$, $P_{j-1} = nbd(P_j)$ in $G_{M_{\ell_j}}$ and argue that ℓ_{j-1} must appear in $\mu(q)$ for each $q \in P_{j-1}$. By Proposition A.5, $P_{j-1} \setminus P_j \neq \emptyset$ and $P_k \subset P_{k-1} \subset \cdots \subseteq P$. Recall that $|P_k| \geq 2$, since $p \in P_k$ as well as the witness q such that $q?p \in M_{\ell_k}$. Since $|P| = k$, we must thus have $P_2 = P$. In other words, ℓ_1 appears in $\mu(q)$ for each $q \in P_2 = P$. From Definition A.6, it follows that ℓ_0 is dead in (μ, ν, τ). □ □

Let (μ, ν, τ) be a state of $\mathcal{A}^{\circledast}$. For any process p and any $P \subseteq \mathcal{P}$, there are at most $|P|$ live labels in $\mu(p)$ of type P. Thus, the number of live labels in $\mu(p)$ is bounded by $|\mathcal{P}| \cdot 2^{|\mathcal{P}|}$ and the number of live labels overall in (μ, ν, τ) is bounded by $|\mathcal{P}|^2 \cdot 2^{|\mathcal{P}|}$.

A finite-state version of $\mathcal{A}^{\circledast}$ From this, we can derive a finite-state version of $\mathcal{A}^{\circledast}$ when the language accepted by $\mathcal{A}$ is com-connected. Instead of using the infinite set of labels $\mathbb{N}$ to name factors, we fix a finite set of labels Γ such that $|\Gamma| > |\mathcal{P}|^2 \cdot 2^{|\mathcal{P}|}$. Thus, a state of $\mathcal{A}^{\circledast}$ now consists of functions (μ, ν, τ) where $\mu : \mathcal{P} \to \Gamma^*$, $\nu : \Gamma \to Conf_{\mathcal{A}}$ and $\tau : \Gamma \to 2^{\mathcal{P}}$.

A state of $\mathcal{A}^{\circledast}$ is a triple (μ, ν, τ) that satisfies the following conditions:

- For any pair of processes p and q (not necessarily distinct) and any pair of distinct labels ℓ and ℓ', if ℓ appears before ℓ' in $\mu(p)$, then ℓ' does not appear before ℓ in $\mu(q)$.
- If $\tau(\ell) \neq \emptyset$ then ℓ appears in $\mu(p)$ for some $p \in \mathcal{P}$. Moreover, if ℓ appears in $\mu(p)$ then $p \in \tau(\ell)$.
- For each $p \in \mathcal{P}$, $\mu(p)$ contains at most $|P|$ labels of type P for each $P \subseteq \mathcal{P}$.

The last condition ensures that $\mathcal{A}^{\circledast}$ is finite-state.

In the initial state $(\mu_{in}, \nu_{in}, \tau_{in})$, $\mu_{in}(p) = \varepsilon$ for each $p \in \mathcal{P}$, $\nu_{in}(\ell) = (s_{in}, \chi_\varepsilon)$ for each $\ell \in \Gamma$ and $\tau(\ell) = \emptyset$ for each $\ell \in \Gamma$.

Let (μ, ν, τ) and (μ', ν', τ') be two states of $\mathcal{A}^{\circledast}$ and let $a \in \Sigma_p$. Then $(\mu, \nu, \tau) \xrightarrow{a}{}' (\mu', \nu', \tau')$ provided we can construct an intermediate triple of functions (μ'', ν'', τ'') such that:

- For $q \neq p$, $\mu''(q) = \mu(q)$.
- Either $\mu''(p) = \mu(p)$ or $\mu''(p) = \mu(p) \cdot \ell$ for some $\ell \in \Gamma$ such that ℓ does not already appear in $\mu(p)$.
- Let the last label in $\mu''(p)$ be ℓ. Then, $\nu(\ell) \overset{a}{\Longrightarrow} \nu''(\ell)$ and for $\ell' \neq \ell$, $\nu''(\ell') = \nu(\ell')$.
- Let the last label in $\mu''(p)$ be ℓ. Then $\tau''(\ell) \supset \{p\}$ and for $\ell' \neq \ell$, $\tau''(\ell') = \tau(\ell')$. Moreover, if ℓ already appears in $\mu(q)$ for some $q \in \mathcal{P}$, then $\tau'(\ell) = \tau(\ell)$.

For $p \in \mathcal{P}$ and $P \subseteq \mathcal{P}$, suppose that $\mu(p)$ is of the form $u_0\ell_0u_1\ell_1 \ldots \ell_k u_k \ell_{k+1} u_{k+1}$, where each u_i, $i \in \{0, 1, \ldots, k+1\}$, is a string over Γ, $\tau(\ell_0) = \tau(\ell_1) = \cdots = \tau(\ell_{k+1}) = P$ and $|P| = k$.

Then, it is the case that ℓ_0 is dead in (μ'', ν'', τ'') and $\nu''(\ell_0)$ is a final configuration of $\mathcal{A}$. (Observe that since exactly one process moves on each input, at most one dead label is generated with each move).

- (μ', ν', τ') is obtained from (μ'', ν'', τ'') by deleting the dead label ℓ_0, if any, from $\mu(q)$ for each $q \in \tau''(\ell_0)$ and then resetting $\tau'(\ell_0) = \emptyset$ and $\nu'(\ell_0) = (s_{in}, \chi_\varepsilon)$. If there are no dead labels in (μ'', ν'', τ''), then (μ', ν', τ') is the same as (μ'', ν'', τ'').

A state (μ, ν, τ) of $\mathcal{A}^{\circledast}$ is in F' provided:

- If ℓ appears in $\mu(p)$ for some process p, $\nu(\ell)$ is a final configuration of $\mathcal{A}$.
- If ℓ does not appear in $\mu(p)$ for any process p, $\nu(\ell) = (s_{in}, \chi_\varepsilon)$.
- If $\tau(\ell) = P$ then ℓ appears in $\mu(p)$ for each $p \in P$.

From Lemma A.7, it is easy to argue that if $\mathcal{L}$ is com-connected, then the finite-state version of $\mathcal{A}^{\circledast}$ accepts $\mathcal{L}^{\circledast}$. This completes the proof of Theorem A.3.