BRICS

Basic Research in Computer Science

BRICS RS-99-52 Henriksen et al.: Towards a Theory of Regular MSC Languages

Towards a Theory of Regular MSC Languages

Jesper G. Henriksen
Madhavan Mukund
K. Narayan Kumar
P. S. Thiagarajan

BRICS Report Series RS-99-52

ISSN 0909-0878 December 1999

See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

http://www.brics.dk
ftp://ftp.brics.dk
This document in subdirectory RS/99/52/

Towards a Theory of Regular MSC Languages

Jesper G. Henriksen
BRICS[*], Dept. of Comp. Sci., University of Aarhus, Denmark
Email: `gulmann@brics.dk`

Madhavan Mukund, K. Narayan Kumar, P. S. Thiagarajan
Chennai Mathematical Institute, Chennai, India
Email: {`madhavan`, `kumar`, `pst`}`@smi.ernet.in`

December, 1999

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 Message Sequence Graph (MSG). We show that the class of languages arising from the so-called locally synchronized MSGs constitute a proper subclass of the languages which are regular in our sense. In fact, we characterize the locally synchronized MSG languages as the subclass of regular MSC languages that are *finitely generated*.

[*] **B**asic **R**esearch **i**n **C**omputer **S**cience,
Centre of the Danish National Research Foundation.

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 [11, 20]. They also appear in the literature as timing sequence diagrams, message flow diagrams and object interaction diagrams and are used in a number of software engineering methodologies [4, 10, 20]. 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 the following: 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.

A standard way to generate a collection of MSCs is to use a Hierarchical (or High-level) Message Sequence Chart (HMSC) [14]. An HMSC is a finite directed graph in which each node is labelled, in turn, by an HMSC. The HMSCs labelling the vertices may not refer to each other. The collection of MSCs represented by an HMSC consists of all MSCs obtained by tracing a path in the HMSC from an initial vertex to a terminal vertex and concatenating the MSCs that are encountered along the path.

Because of the restrictions on the labelling of HMSCs, we can derive an equivalent Message Sequence Graph (MSG) by flattening out the hierarchical labelling in an HMSC. In other words, an MSG is a graph where each node is labelled by a simple MSC. Like an HMSC, an MSG defines a collection of MSCs obtained by concatenating the MSCs labelling 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 restricts one's attention to characterizing structural properties of MSGs rather than of HMSCs [2, 17, 19].

In [2], Alur and Yannakakis study the restricted class of *locally synchronized* (or *bounded*) MSGs. They show that the collection of MSCs generated by a locally synchronized MSG can be represented as a regular string language. As a result, the behaviours captured by a locally synchronized MSG can be, in principle, realized as a finite-state automaton. It is easy to see that not every MSG-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. A second consequence is that locally synchronized MSGs define a strict subclass of regular collections of MSCs. A final consequence is that our notion leads to a state-based representation that is one step closer to an implementation than the description of system requirements using MSGs. Stated differently, our work also addresses the issue, raised in [7], 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 focussing 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 [8, 23, 24]. 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 which usually induces temporal logics using which one can reason about such collections [23]. 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 [25]. 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.

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 labelled partial order in which the labels come from a finite alphabet Σ which 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, unlike the classical setting of strings (or trees or Mazurkiewicz traces), the universe $\mathcal{M}$ is itself *not* regular according to our definition. This fact 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 for a natural number B if at every "prefix" of the MSC and for every pair of processes (p, q) there are at most B messages that p has sent to q which have yet to be received by q. A language of MSCs is B-bounded if every member of the language is B-bounded. Fortunately, for every regular MSC language L we can effectively compute a (minimal) bound B such that L is B-bounded. 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, it is never the case that a channel 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 $\mathrm{MSO}(\mathcal{P}, B)$.

In related work, a number of studies are available which are concerned with individual MSCs in terms of their semantics and properties [1, 12]. 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 [13, 17, 19] and detection of process divergence and non-local choice [3]. A systematic account of the various model-checking problems associated with MSGs and their complexities is given in [2].

In general, the language defined by an MSG is not regular. Conversely, we exhibit a regular MSC language which cannot be represented by an MSG, let alone a locally synchronized MSG. We then characterize the class of regular MSC languages which can be defined by message sequence graphs. We first observe that the MSC languages defined by MSGs are finitely generated. It turns out that there are regular MSC languages which consist of an infinite number of "atomic" MSCs and are hence not finitely generated.

We give a decision procedure to determine when a regular MSC language is finitely generated. Following this, we establish that the class of finitely generated regular MSC languages coincides with the class of languages defined by locally synchronized MSGs. In one direction, this characterization hinges crucially on elements of Mazurkiewicz trace theory [8, 26].

In this paper we confine our attention to *finite* MSCs. We feel however that our results will serve as a good launching pad for a similar account concerning infinite MSCs. This should then lead to the design of appropriate temporal logics and automata-theoretic solutions (based on message-passing automata) to model-checking problems for these logics.

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 one basic result and a couple of proof techniques from the theory of Mazurkiewicz traces [8]. However, we need to modify some of 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 labelled 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 locally synchronized MSG languages.

2 Regular MSC Languages

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite set of processes (or agents) which 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. At our level of abstraction, we shall not be concerned with the actual messages that are sent and received. We will also not deal with the internal actions of the agents. We set $\Sigma = \bigcup_{p \in \mathcal{P}} \Sigma_p$ and let a, b range over Σ. We also denote the set of *channels* by $Ch = \{(p, q) \mid p \neq q\}$ and let c, d range over Ch.

A Σ-labelled poset is a structure $M = (E, \leq, \lambda)$ where $(E, \leq)$ is a poset and $\lambda : E \rightarrow \Sigma$ 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 $c \in Ch$, we define the relation $R_c \stackrel{\text{def}}{=} \{(e, e') \mid \lambda(e) = p!q, \lambda(e') = q?p \text{ and } |\downarrow e \cap E_{p!q}| = |\downarrow e' \cap E_{q?p}|\}$. Since messages are assumed to be read in FIFO fashion, $e \; R_{(p,q)} \; e'$ 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 $R_p \stackrel{\text{def}}{=} (E_p \times E_p) \cap \leq$.

An MSC (over $\mathcal{P}$) is a *finite* Σ-labelled poset $M = (E, \leq, \lambda)$ which satisfies the following conditions[1]:

(1) Each R_p is a linear order.

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

(3) $\leq \; = (R_{\mathcal{P}} \cup R_{Ch})^*$ where $R_{\mathcal{P}} = \bigcup_{p \in \mathcal{P}} R_p$ and $R_{Ch} = \bigcup_{c \in Ch} R_c$.

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

[1]Our definition captures the standard partial-order semantics associated with MSCs in, for instance, [1, 20].

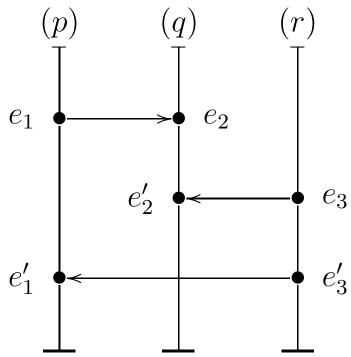

Figure 1: An example MSC over $\{p, q, r\}$.

the relation R_{Ch} are displayed as horizontal or downward-sloping directed edges. 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 the later events. The R_{Ch}-edges across agents are depicted by horizontal edges—for instance $e_3 \; R_{(r,q)} \; e'_2$. The labelling function λ is easy to extract from the diagram—for example, $\lambda(e'_3) = r!p$ and $\lambda(e_2) = q?p$.

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 $\mathcal{L} \subseteq \mathcal{M}(\mathcal{P})$. From now on, whenever there is no confusion we omit $\mathcal{P}$ and denote $\mathcal{M}(\mathcal{P})$ by $\mathcal{M}$ and $\Sigma_{\mathcal{P}}$ by Σ.

We define regular MSC languages in terms of their linearizations. For an MSC $M = (E, \leq, \lambda)$, we let $lin(M) \stackrel{\text{def}}{=} \{\lambda(\pi) \mid \pi \text{ is a linearization of } (E, \leq)\}$. 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 $lin(\mathcal{L}) = \bigcup\{lin(M) \mid M \in \mathcal{L}\}$. 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, 18, 19]) 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.

To directly characterize the subsets of Σ^* that correspond to MSC languages, we proceed as follows. Let $Com = \{(p!q, q?p) \mid (p, q) \in Ch\}$. We say that $\sigma \in \Sigma^*$ is *proper* if for every prefix τ of σ and every pair $(a, b) \in Com$, $|\tau|_a \geq |\tau|_b$. We say that σ is *complete* if σ is proper and $|\sigma|_a = |\sigma|_b$ for every $(a, b) \in Com$. Next we define a *context-sensitive* independence relation $I \subseteq \Sigma^* \times (\Sigma \times \Sigma)$ as follows: $(\sigma, a, b) \in I$ if σab is proper, $a \in \Sigma_p$ and $b \in \Sigma_q$ for distinct processes p and q, and if $(a, b) \in Com$ then $|\sigma|_a > |\sigma|_b$. Observe that if $(\sigma, a, b) \in I$ then $(\sigma, b, a) \in I$.

Let $\Sigma^\circ = \{\sigma \mid \sigma \in \Sigma^* \text{ and } \sigma \text{ is complete}\}$. We then 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 $\sigma \sim \sigma'$. 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°.

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

Just as a trace can be identified with its linearizations in Mazurkiewicz trace theory [8], we can identify each MSC with the set of 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 sketch briefly that these maps are "inverses" of each other.

We first define $\mathsf{sm} : \Sigma^\circ \to \mathcal{M}$. Let $\sigma \in \Sigma^\circ$. Then $\mathsf{sm}(\sigma) = (E, \leq, \lambda)$, where

- $E = \{\tau a \mid \tau a \in \mathrm{prf}(\sigma)\}$, where $\mathrm{prf}(\sigma)$ is the set of prefixes of σ. Thus $E = \mathrm{prf}(\sigma) - \{\varepsilon\}$.

- $\leq \, = (R_{\mathcal{P}} \cup R_{Ch})^*$ where $R_{\mathcal{P}} = \bigcup_{p \in \mathcal{P}} R_p$, $R_{Ch} = \bigcup_{c \in Ch} R_c$. The constituent 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 $\tau a \in \mathrm{prf}(\tau' b)$. Moreover, for each $c \in Ch$, $(\tau a, \tau' b) \in R_c$ iff $a = p!q$ and $b = q?p$ for some $p, q \in \mathcal{P}$ and furthermore $|\tau a|_a = |\tau' b|_b$.

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

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'.

Conversely, we define the map $\mathsf{ms} : \mathcal{M} \to \Sigma^\circ/\!\sim$ as: $\mathsf{ms}(M) = lin(M)$ and it is not hard to show that ms is well-defined. We can also show that for every $\tau \in \Sigma^\circ$, $\mathsf{ms}(\mathsf{sm}(\sigma)) = [\sigma]_\sim$ and for every $M \in \mathcal{M}$, $\mathsf{sm}(\mathsf{ms}(M)) = M$. This justifies our claim that $\Sigma^\circ/\!\sim$ and $\mathcal{M}$ are two equivalent ways of representing the same class of objects. Hence, abusing terminology, we will write "MSC language" to mean "string MSC language". From the context, it should be clear whether we are working with MSCs from $\mathcal{M}$ or complete strings over Σ^*. As a rule of thumb, we will use $\mathcal{L}$ to denote the former and L to denote the latter, but this distinction is not always firm.

We can now finally define our notion of a regular collection of MSCs. We will say that $\mathcal{L} \subseteq \mathcal{M}$ is a *regular MSC language* if the corresponding *string* MSC language is a regular subset of Σ^*. Thus, a language $\mathcal{L}$ of MSCs is regular in case $lin(\mathcal{L})$ is regular in the classical sense. Note that, unlike the settings of strings (or trees or Mazurkiewicz traces), the universe $\mathcal{M}$ is itself *not* regular according to our definition. This fact has a strong bearing on the automata-theoretic and logical formulations in our work, as will become apparent later.

Given a regular subset $L \subseteq \Sigma^*$, we can decide whether L is a regular MSC language. We say that a state s in a finite-state automaton is *live* if there is a path from s to a final state. Let $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$ be the minimal DFA representing L. Then 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 $\mathcal{K}_s : Ch \to \mathbb{N}$ which 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 $(a, b) \notin Com$ or $\mathcal{K}_s((p,q)) > 0$, there exists s_1' such that $\delta(s, b) = s_1'$ and $\delta(s_1', a) = s_2$.

Item (4) has useful consequences. As usual, we extend δ to words and let $\delta(s_{in}, u)$ denote the (unique) state reached by $\mathcal{A}$ on reading an input u. Let u be a proper word and let a, b be communication actions such that (u, a, b) belongs to the context-sensitive independence relation defined earlier. Item (4) guarantees that $\delta(s_{in}, uab) = \delta(s_{in}, uba)$. From this, we can conclude that if v, w are complete words such that $v \sim w$, then $\delta(s_{in}, v) = \delta(s_{in}, w)$.

These conditions can be checked in time linear in the size of δ. 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 complete word σ is B-bounded if for each prefix τ of σ and for each channel $(p,q) \in Ch$, $|\tau|_{p!q} - |\tau|_{q?p} \leq B$. We say that $L \subseteq \Sigma^\circ$ is B-bounded if every word $\sigma \in L$ is B-bounded. Let L be a regular MSC language and let $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$

be its minimal DFA, as described above, with capacity functions $\{\mathcal{K}_s\}_{s\in S}$. Let $B_L \stackrel{\text{def}}{=} \max_{s\in S, c\in Ch} \mathcal{K}_s(c)$. Then it is easy to see that L is B_L-*bounded* and that B_L can be effectively computed from $\mathcal{A}$. In particular, we have:

Proposition 2.1 *Let L be a regular MSC language. There is a bound $B \in \mathbb{N}$ such that L is B-bounded.*

Finally, we shall say that the MSC M is B-bounded if every string in $lin(M)$ is B-bounded. A collection of MSCs is B-bounded if every member of the collection is B-bounded.

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 which 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.

$\square$

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).

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* which specifies the queue of messages currently residing in each channel c. The *initial configuration* of $\mathcal{A}$ is $(s_{in}, \chi_\varepsilon)$ where $\chi_\varepsilon(c)$ is the empty string ε for every channel c. The set of *final configurations* of $\mathcal{A}$ is $F \times \{\chi_\varepsilon\}$.

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 $(s, \chi) \overset{p!q}{\Longrightarrow} (s', \chi')$ 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 $(s, \chi) \overset{p?q}{\Longrightarrow} (s', \chi')$ 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 $\tau a \in \mathrm{prf}(\sigma)$, $\rho(\tau) \overset{a}{\Longrightarrow} \rho(\tau a)$. The run ρ is *accepting* if $\rho(\sigma)$ is a final configuration. We define $L(\mathcal{A}) \overset{\text{def}}{=} \{\sigma \mid \mathcal{A} \text{ has an accepting run over } \sigma\}$. 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 $\sigma' \in L(\mathcal{A})$.

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

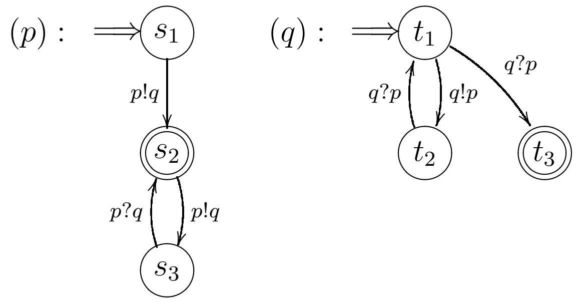

Figure 2: A 3-bounded message-passing automaton.

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 $|\chi(c)| \leq B$. 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 $\mathcal{L} = \{M_i\}_{i=0}^{\omega}$, where M_i is displayed in Figure 3 for $i = 2$.

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].

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

This result follows from the definitions and it constitutes the easy half of the characterization we wish to obtain. 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 for L yields a bound B such that L is B-bounded. We first view L as a regular Mazurkiewicz trace language and apply Zielonka's theorem [26] to obtain a so-called asynchronous automaton $\mathcal{Z}$ for L. We then convert $\mathcal{Z}$ into the desired B-bounded message-passing automaton $\mathcal{A}$ with the property $L(\mathcal{A}) = L$.

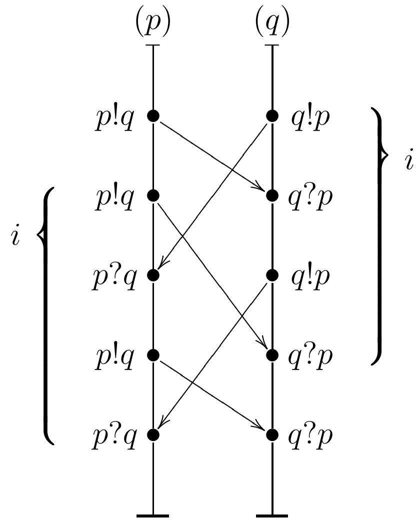

Figure 3: The M_i's accepted by the automaton in Figure 2.

Let $\{\Sigma_c\}_{c \in Ch}$ be given by $\Sigma_c \stackrel{\text{def}}{=} \{p!q, q?p\}$ for $c = (p,q)$. We let $X = \mathcal{P} \cup Ch$. For $a \in \Sigma$, we define the locations of a as $\text{loc}(a) \stackrel{\text{def}}{=} \{x \in X \mid a \in \Sigma_x\}$. The distributed alphabet $\{\Sigma_x\}_{x \in X}$ induces the Mazurkiewicz trace alphabet (Σ, I_X), where $I_X = \{(a,b) \mid a, b \in \Sigma, \text{loc}(a) \cap \text{loc}(b) = \emptyset\}$ is an irreflexive and symmetric independence relation. We can then define $\approx \subseteq \Sigma^* \times \Sigma^*$ to be the least equivalence relation such that if $\sigma = \sigma_1 ab \sigma_2$, $\sigma' = \sigma_1 ba \sigma_2$ and $(a,b) \in I_X$ then $\sigma \approx \sigma'$. We say that $L \subseteq \Sigma^*$ is a *regular (Mazurkiewicz) trace language* over (Σ, I_X) if L is a regular subset of Σ^* and L is $\approx$-closed—that is, for each $\sigma \in \Sigma^*$, if $\sigma \in L$ and $\sigma \approx \sigma'$ then $\sigma' \in L$.

Let $L \subseteq \Sigma^*$ be a regular MSC language. It is not difficult to verify that L is a regular trace language over (Σ, I_X)—the independence relation I_X corresponds to the static (context-insensitive) kernel of the context-sensitive independence relation I defined in Section 2.

In order to apply Zielonka's theorem we need to first introduce asynchronous automata. An asynchronous automaton over the distributed alphabet $\{\Sigma_x\}_{x \in X}$ is a structure $\mathcal{Z} = (\{S_x\}_{x \in X}, \{\longrightarrow_a\}_{a \in \Sigma}, s_{in}, F)$ where each S_x is a finite set of local states of the component x. Let $S = \prod_{x \in X} S_x$ denote the set of global states of $\mathcal{Z}$. Then $s_{in} \in S$ is the global initial state and $F \subseteq S$ is the set of global final states. Let $a = p!q$. Then $\longrightarrow_a \subseteq (S_p \times S_{(p,q)}) \times (S_p \times S_{(p,q)})$. The pair $((s_1, s_1'), (s_2, s_2')) \in \longrightarrow_a$ denotes the fact that the p-component in state s_1, and the channel (p,q)-component in state s_1' can together execute $p!q$ and move to the joint state (s_2, s_2'). Similarly, for a receive action $b = q?p$,

$\longrightarrow_b \subseteq (S_{(p,q)} \times S_q) \times (S_{(p,q)} \times S_q)$ defines joint moves of the channel (p,q) and process q when q receives messages from p. To define the global transition relation $\longrightarrow \subseteq S \times \Sigma \times S$, we let s_x denote the xth component of the global state s. Suppose $s, s' \in S$, $a = p!q$ and $c = (p,q)$. Then $(s, a, s') \in \longrightarrow$ if $((s_p, s_c), (s'_p, s'_c)) \in \longrightarrow_a$ and $s_x = s'_x$ for every $x \in X \setminus \{p, c\}$. Transitions of the form (s, b, s') with $b = q?p$ are defined in a similar fashion. The notions of runs and accepting runs are formulated in the obvious way. We let $L(\mathcal{Z})$ be the subset of Σ^* accepted by $\mathcal{Z}$.

Zielonka's theorem [26] asserts that from a regular trace language L, we can construct a deterministic asynchronous automaton $\mathcal{Z}$ such that $L(\mathcal{Z}) = L$. We have already observed that a regular MSC language $L \subseteq \Sigma^*$ is a regular trace language over (Σ, I_X). It follows that from a regular MSC language $L \subseteq \Sigma^*$ we can effectively construct a deterministic asynchronous automaton $\mathcal{Z}$ over the distributed alphabet $\{\Sigma_x\}_{x \in X}$ such that $L = L(\mathcal{Z})$.

Fix a B-bounded regular MSC language L and let $\mathcal{Z} = (\{S_x\}_{x \in X}, \{\longrightarrow_a\}_{a \in \Sigma}, s_{in}, F)$ be a deterministic asynchronous automaton such that $L = L(\mathcal{Z})$. We claim that we can effectively transform $\mathcal{Z}$ into a B-bounded message-passing automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L$.

This transformation is complicated by the following fact. In $\mathcal{Z}$, for each pair $p, q \in \mathcal{P}$, the actions $p!q$ and $q?p$ are performed by the channel component (p,q) and are hence dependent on each other in all contexts. As a result, the transition relations $\longrightarrow_{p!q}$ and $\longrightarrow_{q?p}$ do not reflect the context-sensitive independence of the actions $p!q$ and $q?p$, even though the language accepted by $\mathcal{Z}$ is a regular MSC language and is hence closed with respect to the context-sensitive independence relation I on Σ. This means that for two inputs σ and σ' such that $\sigma \sim \sigma'$, $\mathcal{Z}$ will, in general, admit drastically different runs on σ and σ'. On the other hand, the structure of message-passing automata is such that the moves of any message-passing automaton over Σ can be reordered with respect to the independence relation I. This implies that the simulation of $\mathcal{Z}$ by $\mathcal{A}$ should not depend on the order in which independent occurrences of actions of the form $p!q$ and $q?p$ are linearized in a given input.

To get around this, we simulate the component (p,q) of $\mathcal{Z}$ in $\mathcal{A}$ using the components p and q such that for each input σ, p and q keep track the moves of (p,q) along a *canonical* reordering $\sigma' \sim \sigma$. This simulation is coordinated using the messages sent from p to q.

The key technical input for this simulation comes from [15] where it is shown how each process p in a message-passing system can use a bounded time-stamping protocol to keep track of the latest information about every other process in the system. The protocol does not add extra messages to the system. This protocol also allows each process p to locally keep track of the messages sent on each channel (p, q) for which p has not received an "acknowledgment", directly or indirectly, from q. This list of "unacknowledged" messages yields an upper bound for the number of messages currently resident in each outgoing channel from p. (A more detailed description of the time-stamping protocol is presented in Appendix A.)

The desired automaton $\mathcal{A}$ will be of the form $(\{\mathcal{A}'_p\}_{p\in\mathcal{P}}, \Delta, s'_{in}, F')$, where $\mathcal{A}'_p = (S'_p, \rightsquigarrow_p)$. For each process p, each state in S'_p is of the form $\langle s_p, \bar{s}_p, \tau_p\rangle$ where s_p records a local state of p in $\mathcal{Z}$, $\bar{s}_p$ records a local state s_c in $\mathcal{Z}$ for each incoming channel $c = (q, p)$ at p, and τ is a time-stamp generated by protocol of [15].

The message alphabet is $\Delta = Ev \times \mathcal{T}$ where $Ev = \bigcup_{a\in\Sigma} \longrightarrow_a$ and $\mathcal{T}$ is the set of time-stamps used by the protocol of [15]. (Recall that $\longrightarrow_a$ is the set of a-transitions specified in $\mathcal{Z}$ for each a.) The initial state and the final states are defined in the expected manner using the initial and final states of $\mathcal{Z}$.

The transitions of $\mathcal{A}$ are arranged as follows. The tuple $(\langle s_p, \overline{s}_p, \tau_p\rangle, p!q, (e, \tau), \langle s'_p, \overline{s'}_p, \tau'_p\rangle)$ belongs to the p-local transition relation $\rightsquigarrow_p$ provided the following hold. First, $\tau = \tau'_p$ and τ'_p is the time-stamp generated from τ_p by the protocol of [15]. Let $c = (p, q)$. The e-component of the message is a move $((s_p, s_c), (s'_p, s'_c)) \in \longrightarrow_{p!q}$ for some $s_c, s'_c \in S_c$. Finally, according to τ_p there are at most $B-1$ "unacknowledged" messages in the channel c, indicating that sending this message will not violate the B-boundedness of $\mathcal{A}$.

The tuple $(\langle s_p, \overline{s}_p, \tau_p\rangle, p?q, (e, \tau'), \langle s'_p, \overline{s'}_p, \tau'_p\rangle)$ belongs to $\rightsquigarrow_p$ provided the following hold. From the time-stamp τ' on the incoming message, p collects the latest information from each process $r \in \mathcal{P}$ about new $r!p$ events that have been sent by r but not yet received by p. For each such event, the time-stamp τ' also records the move $((s_r, s_c), (s'_r, s'_c))$ guessed by r when the event occurred. Process p updates the (r, p)-component of $\overline{s}_p$ by applying this move guessed by r. If this guess is not permitted by the current state of (r, p) as recorded in $\overline{s}_p$, p gets stuck. If there is more than one such $r!p$ event then p processes each of them in the order in which they were sent. Let the resulting states corresponding to the channel components $\{(r, p) \mid r \in \mathcal{P}\}$ be $\hat{s}_p$. Let $c = (q, p)$. Now, p simulates the unique $p?q$ move $(\hat{s}_c, s_p) \xrightarrow{p?q} (\overline{s}'_c, s'_p)$ of $\mathcal{Z}$ (recall that $\mathcal{Z}$ is deterministic). With this, p has updated the components s_p and $\bar{s}_p$ of its state to s'_p and $\overline{s}'_p$. Finally, it uses the time-stamps τ_p and τ' to generate a new time-stamp τ'_p as specified by the protocol of [15].

It is easy to show that $L(\mathcal{Z}) \subseteq L(\mathcal{A})$. To show the converse, let $\sigma \in L(\mathcal{A})$ and let ρ be an accepting run of $\mathcal{A}$ over σ. From the way $\mathcal{A}$ simulates $\mathcal{Z}$, we can show that there is a canonical reordering $\sigma' \sim \sigma$ such that there is an accepting run ρ' of $\mathcal{A}$ over σ' where ρ' is just a reordered version of ρ. The word σ' has the property that each message is received as soon as possible, subject to causality constraints. For instance, if $\sigma = p!q\ p!q\ q?p\ q?p$ then $\sigma' = p!q\ q?p\ p!q\ q?p$, and if $\sigma = p!q\ p!q\ p!r\ r?p\ r!q\ q?r\ q?p\ q?p$, then $\sigma' = \sigma$. In the second example, the messages via r ensure that p will have sent *both* messages to q before q receives the first one. From ρ' it is easy to extract an accepting run of $\mathcal{Z}$ over σ'. The language accepted by $\mathcal{Z}$ is $\sim$-closed because L is $\sim$-closed. Consequently, σ is also accepted by $\mathcal{Z}$.

Filling in the details of this proof skeleton leads to the following result.

Lemma 3.3 *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 $L(\mathcal{A}) = L$.*

We say that $\mathcal{A}$ is a bounded message-passing automaton if $\mathcal{A}$ is B-bounded for some $B \in \mathbb{N}$. The main result of this section is an easy consequence of the previous result.

Theorem 3.4 *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 $L(\mathcal{A}) = L$.*

The automaton $\mathcal{A}$ constructed above is nondeterministic because each send-action requires guessing a move of $\mathcal{Z}$. In [16], a direct construction is presented to generate a deterministic bounded message-passing automaton for each regular MSC language.

4 A Logical Characterization

We formulate a monadic second-order logic which 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 which assigns to each individual variable 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 the techniques used for proving a similar result in the theory of traces [9] 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 $L = \bigcup\{lin(M) \mid M \in \mathcal{L}_\varphi\}$. 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 [5]. 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:

$$(\exists X_{\mathcal{K}_0})(\exists X_{\mathcal{K}_1}) \cdots (\exists X_{\mathcal{K}_n})(COMP \wedge \|\varphi\|)$$

where $COMP$ and $\|\varphi\|$ are defined as follows. We provide these definitions in textual form to enhance readability. They can be easily converted to formulas in $\mathrm{MSO}(\Sigma)$.

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 $M = (E, \leq, \lambda)$, $\leq = (R_{\mathcal{P}} \cup R_{Ch})^*$.

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). 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* $\mathrm{MSO}(\mathcal{P}, B)$.

Let $L = \bigcup\{lin(M) \mid M \in \mathcal{L}\}$. Then L is a regular (string) MSC language over Σ. Hence by Büchi's theorem [5] 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 $\widehat{\varphi}$ which 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 [22, Lemma 15], which asserts an identical result in the setting of (infinite) Mazurkiewicz traces. □

We next exhibit a formula in MSO($\mathcal{P}, B$) (for any $B \in \mathbb{N}$) which 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:

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

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 $L = \bigcup\{lin(M) \mid M \in \mathcal{L}\}$. Then L is a regular (string) MSC language over Σ. Hence by Büchi's theorem

[5] there exists a sentence φ in MSO(Σ) such that $L = \{\sigma \mid \sigma \models \varphi\}$. We now define the formula $\widehat{\varphi} = \|\varphi\|$ in 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 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 MSO(Σ) is decidable, it follows that 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 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 flatten out 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 which 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.

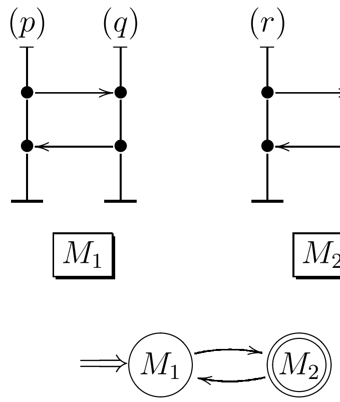

Figure 4: An example MSG.

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 for MSCs such that E_1 and E_2 are disjoint. For $i \in \{1, 2\}$, let R^i_c and $\{R^i_p\}_{p\in\mathcal{P}}$ denote the underlying communication and process causality relations in M_i. 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 $\leq = (R_{\mathcal{P}} \cup R_{Ch})^*$, where $R_p = R^1_p \cup R^2_p \cup \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2, \lambda(e_1) \in \Sigma_p, \lambda(e_2) \in \Sigma_p\}$ for $p \in \mathcal{P}$, and $R_c = R^1_c \cup R^2_c$ for $c \in Com$.

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 $\mathcal{L}(\mathcal{G}) = \{M(\pi) \in \mathcal{M} \mid \pi \text{ is a run through } \mathcal{G}\}$.

An example of an MSG is depicted in Figure 4. 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 arbitrary projections.)

A number of studies are available which are concerned with individual MSCs in terms of their semantics and properties [1, 12]. The rest of the work in the area consists of checking specific properties of communication scenarios specified as MSGs. We briefly survey these results here.

Muscholl, Peled, and Su [19] investigate various *matching* problems for MSCs and MSGs, where matching denotes embedding one partial order in another. More specifically, they show that given MSGs $\mathcal{G}_1$ and $\mathcal{G}_2$, it is NP-complete to decide whether there exist MSCs $M_1 \in \mathcal{L}(\mathcal{G}_1)$ and $M_2 \in \mathcal{L}(\mathcal{G}_2)$ such that M_1 matches M_2. The universal counterpart to this problem—that is, does there exists some $M_1 \in \mathcal{L}(\mathcal{G}_1)$ such that M_1 matches every $M_2 \in \mathcal{L}(\mathcal{G}_2)$—is also NP-complete.

Muscholl [17] defines "and-or" versions of MSGs, reminiscent of alternating automata. Given an "and-or" MSG $\mathcal{G}_1$ and a conventional MSG $\mathcal{G}_2$, she considers the problem of deciding whether $\mathcal{G}_1$ admits a run-tree such that each MSC generated by a path in this run-tree matches an MSC in $\mathcal{L}(\mathcal{G}_2)$. She shows that this problem is PSPACE-complete, and moreover that a similar problem of matching LTL-definable properties and MSGs is PSPACE-complete as well.

In [3] Ben-Abdallah and Leue identify and characterize two properties which are intuitively undesirable from an implementation point of view and give algorithms to detect such anomalies. The first of them is that of *process divergence* signifying that the specification allows some process to have an unbounded number of unreceived messages in its buffer. This can be detected in time linear in the total number of messages in the specification. We will note here that though related, the notion of divergence-freeness is implied by our notion of regularity, but does *not* coincide with it. Figure 4 provides a simple counter-example, because the language of the MSG is divergence-free but not regular.

The second underspecification detected by Ben-Abdallah and Leue is that of *nonlocal choice*. Intuitively, this denotes the existence of branching choices where different processes have the possibility of taking conflicting routes in the MSG specification. To prevent such consistency problems, additional messages or history variables have to be introduced into the system. Thus, it is desirable for an MSG to not permit nonlocal choice. Once again, Ben-Abdallah and Leue give an algorithm to detect the existence of nonlocal choice in an MSG which runs in time linear in the total number of messages in the specification. We note that the language of an MSG $\mathcal{G}$ might be a regular MSC language and still exhibit nonlocal choice, as the two notions are not related to each other. It is also worth pointing out that the problem

$$
\begin{tikzcd}
p \arrow[r] & q \\
r \arrow[u] \arrow[ur] &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
p \arrow[r, bend left] & q \arrow[l, bend left] \\
r \arrow[r, bend left] & s \arrow[l, bend left]
\end{tikzcd}
$$

Figure 5: CG_M of Figure 1 (left) and $CG_{M_1 \circ M_2}$ of Figure 4 (right).

of nonlocal choice is limited to specification formalisms such as MSGs and is not an issue in specifications based on message-passing automata.

Alur and Yannakakis [2] consider model checking problems for systems modeled as MSGs with respect to various semantics. In their setup, automata are used to describe undesirable linearizations. Thus, the property to be checked captures the *complement* of the intended behaviour. In this framework, the decision problems essentially reduce to checking emptiness for products of automata, as the constituent automata need not be complemented. They show that for synchronous concatenation of MSCs on the paths of the MSG, the problem is coNP-complete, while the problem is undecidable in general for asynchronous concatenation.

Following this negative result, they then define the notion of a *locally synchronized* MSG. 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 then 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.

The MSG $\mathcal{G}$ is *locally synchronized*[2] 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 5 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.

Interestingly, Alur and Yannakakis [2] then show that the asynchronous model checking problem becomes PSPACE-complete for *locally synchronized* MSGs. Clearly, the MSG of Figure 4 is *not* locally synchronized. This is no coincidence, as it follows as a corollary of their proof sketch [2, Thm. 7] that every locally synchronized MSG-language is indeed regular.

[2]This notion is called “bounded” in [2]. The terminology “locally synchronized” is taken from [18].

We conclude this section by pointing out that Muscholl and Peled [18] also consider two decision problems related to locally synchronized MSGs. The first such problem is that of checking for *race conditions*. This essentially consists of checking whether the causal order allows more linearizations than the visual order. Recalling our discussion of linearizations in Section 2 we see that in Figure 1 there is a race on process q between the receive events for the messages from p and r, respectively.

The other problem is to detect *confluence* of MSG specifications. An MSG $\mathcal{G}$ is said to be confluent in case for any two prefixes M_1, M_2 of MSCs in $\mathcal{L}(\mathcal{G})$ that are consistent (in the sense that both are prefixes of some common MSC), there does indeed exist such a completed MSC M in $\mathcal{L}(\mathcal{G})$ of which both M_1 and M_2 is a prefix.

Muscholl and Peled show that both the problem of deciding whether an MSG has race conditions and the problem of checking whether it is confluent are undecidable in general. However, they emphasize the importance of locally synchronized MSGs by additionally proving that both problems are EXPSPACE-complete for *locally synchronized* MSGs.

6 Finitely Generated Regular MSC Languages

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 constituting 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 which 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 4 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 $M_1, M_2 \in \mathcal{M}$ such that $M = M_1 \circ M' \circ M_2$. 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 $\mathit{Atoms}(M)$ denote the set $\{M' \mid M' \text{ is an atom and } M' \text{ is a component of } M\}$. For an MSC language $\mathcal{L} \subseteq \mathcal{M}$, $\mathit{Atoms}(\mathcal{L}) = \bigcup\{\mathit{Atoms}(M) \mid M \in \mathcal{L}\}$. It is clear that the question of deciding whether $\mathcal{L}$ is finitely generated is equivalent to that of checking whether $\mathit{Atoms}(\mathcal{L})$ 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 which together accept the linearizations of the MSCs in $\mathit{Atoms}(\mathcal{L})$. 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 $\mathcal{K}_s : \mathit{Ch} \to \mathbb{N}$ which 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. The following claims are easy to prove.

Claim 6.2 *Let M be an MSC in $\mathit{Comp}(\mathcal{L})$ (in particular, in $\mathit{Atoms}(\mathcal{L})$) 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 $\mathit{Comp}(\mathcal{L})$, 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 $\mathit{Comp}(\mathcal{L})$. 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 $w = w_1w_2$ 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 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) 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 which 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. □

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 [6] — to show that the asynchronous iteration of a com-connected regular MSC language is regular. The proof in [6] is based on grammars. A more direct, automata-theoretic proof of the same result is described in Appendix B.

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 $\mathcal{A}$ which 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 ([8], 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[3] 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:

[3] A trace is said to be *connected* if, when viewed as a labelled partial order, its Hasse diagram consists of a single connected component. See [8] for a more formal definition.

- $Q = S' \times (Atoms(\mathcal{L}) \cup \{\varepsilon\})$.
- $Q_{in} = \{(s'_{in}, \varepsilon)\}$.
- $(s, b) \longrightarrow (s', b')$ 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 $w = b_1 b_2 \dots b_n b$.

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 w describes a connected trace over $(Atoms(\mathcal{L}), I_A)$. From this, 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 2.1.

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 labelled $p!q$ 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 labelled $q?p$ in $\downarrow e$. This means that $M(\pi)^k$ admits a linearization v'_k with a prefix τ'_k which includes all the events labelled $p!q$ and excludes all the events labelled $q?p$, so that $|\tau|_{p!q} - |\tau|_{q?p} \geq k$.

By Proposition 2.1, 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 $(p, q) \in Ch$, $|\tau|_{p!q} - |\tau|_{q?p} \leq B$. 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 $k \geq 0$,

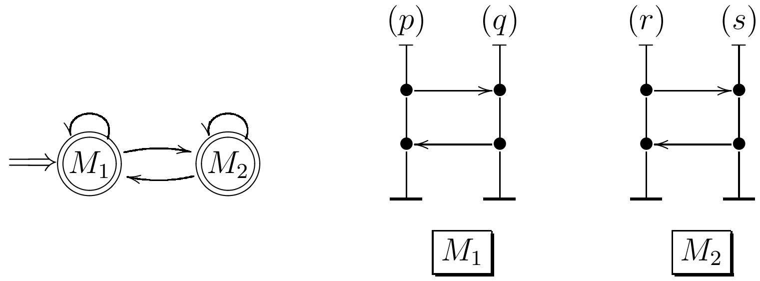

Figure 6: An non-locally synchronized MSG whose language is regular.

$u_k = v_1 v'_k v_2 \in \mathcal{L}$ 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 $|\tau_k|_{p!q} - |\tau_k|_{q?p} \geq B{+}1$, 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 6, 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 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 [21]. 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) [8]. 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 i, we create $1+|A_i|$ processes which 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 $1 \leq$

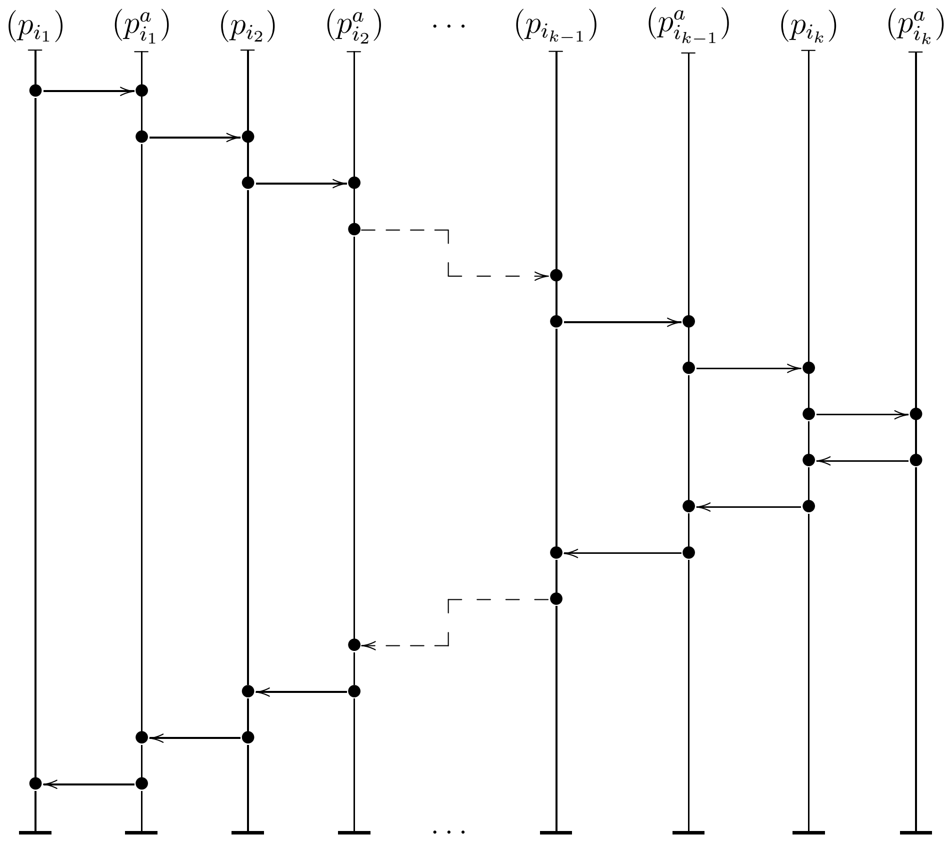

Figure 7: The MSC M_a encoding the letter $a \in A$.

$i_1 < i_2 < \ldots < i_k \leq n$. The MSC M_a representing a is then given in Figure 7. 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. □

7 Discussion

In this report, we have first introduced a notion of regularity for collections of MSCs and provided both automata-theoretic and logical characterizations of regular MSC languages. We can summarize these results as follows.

Theorem 7.1 *Let $L \subseteq \Sigma^*$, where Σ is the communication alphabet associated with a set $\mathcal{P}$ of processes. Then, the following are equivalent.*

(1) L is a regular MSC language.

(2) L is a B-bounded regular MSC language, for some $B \in \mathbb{N}$.

(3) There exists a bounded message-passing automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L$.

(4) L is $\mathrm{MSO}(\mathcal{P}, B)$*-definable, for some $B \in \mathbb{N}$.*

Following this, we have shown that the class of regular MSC languages properly subsumes the class of languages defined by locally synchronized MSGs, as studied in [2]. In fact, we precisely characterize the class of languages definable by locally synchronized MSGs in terms of finitely generated regular MSC languages. One way to phrase our characterization is as follows.

Theorem 7.2 *Let $\mathcal{L}$ be a regular MSC language. Then $\mathcal{L}$ can be described by an MSG if and only if $\mathcal{L}$ is finitely generated.*

Finally, we have also shown that it is decidable whether a given regular MSC language is finitely generated but it is undecidable, in general, whether a given MSG defines a regular MSC language.

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] 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

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

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

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

[7] Damm, W., Harel, D.: LCSs: Breathing life into message sequence charts. *Proceedings of the 3rd IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, Kluwer Academic Publishers (1999) 293–312

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

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

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

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

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

[13] 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

[14] 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

[15] Mukund, M., Narayan Kumar, K., Sohoni, M.: Keeping track of the latest gossip in message-passing systems. *Proceedings of Structures in Concurrency Theory (STRICT)*, Workshops in Computing Series, Springer-Verlag (1995) 249–263

[16] 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, Springer-Verlag (to appear)

[17] 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

[18] 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

[19] 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

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

[21] 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

[22] Thiagarajan, P. S., Walukiewicz, I.: An expressively complete linear time temporal logic for Mazurkiewicz traces. *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*, IEEE Computer Society Press (1997) 183–194

[23] 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

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

[25] 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

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

A Bounded Time-stamps

Partial computations Let $u \in \Sigma^*$ be a complete word and let $\mathsf{sm}(u) = (E_u, \leq_u, \lambda_u)$ be the MSC associated with u. A set of events $I \subseteq E_u$ is called an *(order) ideal* if I is closed with respect to $\leq$—that is, $e \in I$ and $f \leq e$ implies $f \in I$ as well. The ideal $I \subseteq E_u$ is said to be *complete* if I is itself an MSC. We denote the set of ideals of $M = (E, \leq, \lambda)$ by $\mathit{Ideals}(M)$.

Ideals denote consistent partial computations of u—notice that any linearization of an ideal forms a proper word, which we will sometimes refer to as a communication sequence. If an ideal I is complete, then its linearizations are also complete.

We can extend the map sm to the domain of proper words. If u is proper but not complete, $\mathsf{sm}(u)$ will not be an MSC because some send events will not have matching receive events. However, it is easy to observe that the labelled partial order $\mathsf{sm}(u)$ corresponds to an ideal in the MSC $\mathsf{sm}(v)$ for any complete word v which extends u. In this section, we shall work with this extended definition of the map sm. Henceforth, for a proper word u, we shall refer to the labelled partial order $\mathsf{sm}(u)$ as $M_u = (E_u, \leq_u, \lambda_u)$. In particular, E_u always refers to the set of events associated with the MSC $\mathsf{sm}(u)$ generated from a proper word u.

p-views For an ideal I, the $\leq$-maximum p-event in I is denoted $\max_p(I)$, provided there is at least one p-event in I. 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 which p can "see". (By convention, if $\max_p(I)$ is undefined—that is, if there is no p-event in I—the p-view $\partial_p(I)$ is empty.) For $P \subseteq \mathcal{P}$, we use $\partial_P(I)$ to denote $\bigcup_{p \in P} \partial_p(I)$.

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

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

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

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

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

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

The following result is immediate.

Proposition A.1 *Let $u \in \Sigma^*$ be proper. The word u is B-bounded iff for every linearization v of M_u, for every prefix w of v and for every pair of processes p, q, $|w|_{p!q} - |w|_{q?p} \leq B$.*

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

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

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

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

In [15], a protocol is presented for processes to keep track of their primary information during the course of an arbitrary computation.[4] This protocol

[4] In [15], the primary information of an ideal I is defined to include more events than just $\mathit{latest}(I) \cup \mathit{unack}(I)$. However, for our purposes, it suffices to treat events in $\mathit{latest}(I) \cup \mathit{unack}(I)$ as primary.

involves appending a bounded amount of information to each message in the underlying computation, provided the computation is B-bounded. To ensure that the message overhead is bounded, the processes use a distributed time-stamping mechanism which consistently assigns "names" to events using a bounded set of labels.

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

In the protocol of [15], whenever a process p sends a message to q, it first assigns a time-stamp to the new message from a finite set of labels. Process p then appends its primary information to the message being sent. Notice that the current send event will form part of the primary information since it is the latest p-event in $\partial_p(E_u)$. When q receives the message, it can consistently update its primary information to reflect the new information received from p.

The two tricky points in the protocol are for p to decide when it is safe to reuse a time-stamp, and for q to decide whether the information received from p is really new. In order to solve these problems, the protocol of [15] requires processes to also maintain additional time-stamps, corresponding to secondary information. Though we do not need the details of how the protocol works, for completeness we define secondary information.

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 our framework, the protocol of [15] can now be described as follows.

Theorem A.2 *For any $B \in \mathbb{N}$, we can construct a 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 u, $\mathcal{A}^B$ inductively generates a consistent time-stamping τ of E_u. Moreover, for each component $\mathcal{A}_p^B$ of $\mathcal{A}^B$, the local state of $\mathcal{A}_p^B$ at the end of u records the information $\mathit{primary}_p(E_u)$ and $\mathit{secondary}_p(E_u)$ in terms of the time-stamps assigned by τ.*

B 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 [6].

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

Proposition B.1 *Let $L \subseteq \mathcal{M}$ be an MSC language. The MSC $M = (E, \leq, \lambda)$ belongs to $L^{\circledast}$, the* asynchronous iteration *of 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 $M' \in L$.*

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

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

Let 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 $L(\mathcal{A}) = L$. To construct a (sequential) automaton for $L^{\circledast}$, our strategy will be to guess a factorization of the input and simulate $\mathcal{A}$ to verify that each factor belongs to L. We first construct an infinite-state automaton for $L^{\circledast}$ for an arbitrary regular MSC language L and then describe the conditions under which we can restrict the automaton for $L^{\circledast}$ 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 $\nu : \mathbb{N} \to Conf_{\mathcal{A}}$ 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 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, $\nu(\ell) \stackrel{a}{\Longrightarrow} \nu'(\ell)$ 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 B.2 *Let $\mathcal{A}$ be a message-passing automaton for a regular MSC language L. Then, the automaton $\mathcal{A}^{\circledast}$ accepts the language $L^{\circledast}$.*

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 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, $\nu(\ell) \overset{a}{\Longrightarrow} \nu'(\ell)$ 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 which will participate in the factor labelled ℓ and this guess cannot be changed.)

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

B.2 If L is com-connected, $L^{\circledast}$ 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 B.3 *Let L be a regular and com-connected MSC language. Then, $L^{\circledast}$ is regular.*

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

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

Proposition B.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. $\square$

Definition B.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*. $\square$

Lemma B.7 *Let $\mathcal{A}$ be a message-passing automaton for a com-connected MSC language 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 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 B.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 B.6, it follows that ℓ_0 is dead in (μ, ν, τ). $\square$

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 $\mathcal{T}$ such that $|\mathcal{T}| > |\mathcal{P}|^2 \cdot 2^{|\mathcal{P}|}$. Thus, a state of $\mathcal{A}^{\circledast}$ now consists of functions (μ, ν, τ) where $\mu : \mathcal{P} \to \mathcal{T}^*$, $\nu : \mathcal{T} \to \mathit{Conf}_{\mathcal{A}}$ and $\tau : \mathcal{T} \to 2^{\mathcal{P}}$.

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 $\ell \in \mathcal{T}$ and $\tau(\ell) = \emptyset$ for each $\ell \in \mathcal{T}$.

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 $\ell \in \mathcal{T}$.
- Let the last label in $\mu''(p)$ be ℓ. Then, $\nu(\ell) \stackrel{a}{\Longrightarrow} \nu''(\ell)$ 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 $\mathcal{T}$, $\tau(\ell_0) = \tau(\ell_1) = \cdots = \tau(\ell_{k+1}) = P$ 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 $\tau'(\ell_0) = \emptyset$. 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, $\nu(\ell) = s_{in}$.
- If $\tau(\ell) = P$ then ℓ appears in $\mu(p)$ for each $p \in P$.

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

Recent BRICS Report Series Publications

RS-99-52 **Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, and P. S. Thiagarajan.** ***Towards a Theory of Regular MSC Languages*****. December 1999. 43 pp.**

RS-99-51 **Olivier Danvy.** ***Formalizing Implementation Strategies for First-Class Continuations*****. December 1999. Extended version of an article to appear in Smolka, editor,** ***Programming Languages and Systems: Ninth European Symposium on Programming*****, ESOP '00 Proceedings, LNCS 1782, 2000.**

RS-99-50 **Gerth Stølting Brodal and Srinivasan Venkatesh.** ***Improved Bounds for Dictionary Look-up with One Error*****. December 1999. 5 pp.**

RS-99-49 **Alexander A. Ageev and Maxim I. Sviridenko.** ***An Approximation Algorithm for Hypergraph Max k-Cut with Given Sizes of Parts*****. December 1999. 12 pp.**

RS-99-48 **Rasmus Pagh.** ***Faster Deterministic Dictionaries*****. December 1999. 14 pp. Appears in Shmoys, editor,** ***The Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*****, SODA '00 Proceedings, 2000, pages 487–493.**

RS-99-47 **Peter Bro Miltersen and Vinodchandran N. Variyam.** ***Derandomizing Arthur-Merlin Games using Hitting Sets*****. December 1999. 21 pp. Appears in Beame, editor,** ***40th Annual Symposium on Foundations of Computer Science*****, FOCS '99 Proceedings, 1999, pages 71–80.**

RS-99-46 **Peter Bro Miltersen, Vinodchandran N. Variyam, and Osamu Watanabe.** ***Super-Polynomial Versus Half-Exponential Circuit Size in the Exponential Hierarchy*****. December 1999. 14 pp. Appears in Asano, Imai, Lee, Nakano and Tokuyama, editors,** ***Computing and Combinatorics: 5th Annual International Conference*****, COCOON '99 Proceedings, LNCS 1627, 1999, pages 210–220.**

RS-99-45 **Torben Amtoft.** ***Partial Evaluation for Constraint-Based Program Analyses*****. December 1999. 13 pp.**