From Finite State Communication Protocols to High-Level Message Sequence Charts

Anca Muscholl[1] and Doron Peled[2]

[1] LIAFA, Université Paris VII
2, pl. Jussieu, case 7014
F-75251 Paris cedex 05
[2] Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract. The ITU standard for MSCs provides a useful framework for visualizing communication protocols. HMSCs can describe a collection of MSC scenarios in early stages of system design. They extend finite state systems by partial order semantics and asynchronous, unbounded message exchange.
Usually we ask whether an HMSC can be implemented, for instance by a finite state protocol. This question has been shown to be undecidable [6]. Motivated by the paradigm of reverse engineering we study in this paper the converse translation, specifically the question whether a finite state communication protocol can be transformed into an equivalent HMSC. This question also arises when different kinds of specification (HMSC, finite automata, temporal logic) must be integrated into a global one, for instance into an HMSC.
We show in this paper that translating finite state automata into HMSCs is feasible under certain natural assumptions. Specifically, we show that we can test in polynomial time whether a finite state protocol given by a Büchi automaton is equivalent to an HMSC, provided that the automaton satisfies the diamond property (the precise bound is NLOGSPACE-complete). The diamond property is a natural property induced by concurrency. Under the weaker assumption of bounded Büchi automata we show that the test is co-NP-complete. Finally, without any buffer restriction the problem is shown to be undecidable.

Keywords: Message sequence charts, specification, HMSC, bounded automata, partial order specification.

Track B

Contact author: Anca Muscholl
LIAFA, Universite Paris VII
2, pl. Jussieu, case 7014
F-75251 Paris cedex 05
Fax no. +33 1 44 27 68 49
e-mail: `muscholl@liafa.jussieu.fr`

1 Introduction

Message Sequence Charts (MSCs) is a popular formalism used in software development. As other ITU standards, MSC also has a visual notation and therefore it is more appealing than textual formalisms. MSCs are often used in the early design of communication protocols for capturing requirements of a system. They have been known for a longer time as sequence or timing diagrams. A similar formalism is used in UML by describing interactions between objects.

High-level MSCs (HMSC) are used for specifying a collection of MSC scenarios. The standard description consists of a graph where each node contains one MSC. Each maximal path starting from a designated initial state corresponds to a single execution (scenario). Such an execution can be used to denote the communication structure of a typical or an exceptional behavior of a system, or a counterexample found during testing or model checking.

MSCs are based on partial order semantics and do not impose any limitation on buffers (except for fifo message exchange). Therefore, HMSCs are infinite state systems and some verification problems such as model-checking [2] or race conditions [10] are undecidable. One possible solution is to restrict HMSCs in a syntactic way in order to get a finite state space. To this purpose, bounded HMSCs have been proposed in [2, 10]. However, by the partial order semantics automatic verification is expensive for bounded HMSCs. For instance, positive model-checking (inclusion of HMSCs) is complete for exponential space [9, 10].

Using HMSCs as (a possibly incomplete) specification means that they are part of a stepwise refining process until the final implementation stage. So one possible question is whether an HMSC can be implemented, for example by a finite state machine. However, this question turns out to be undecidable [6]. Motivated by the paradigm of reverse engineering we study in this paper the converse translation, specifically the question whether a finite state communication protocol can be transformed into an equivalent HMSC. This form of translation becomes also important when combining different kinds of specification (HMSC, finite automata, temporal logic) into a single one. Since HMSC is a standardized notation, it is reasonable to try to synthesize an HMSC, whenever possible.

The translation from finite automata to HMSCs is well justified by the computational results obtained in this paper. We show that translating finite automata into HMSCs is feasible under certain natural assumptions. Specifically, we show that we can test in polynomial time whether a finite state protocol given by a Büchi automaton is equivalent to an HMSC, provided that the automaton satisfies the diamond property (the precise bound is NLOGSPACE-complete). The diamond property is a natural property induced by concurrency. Under the weaker assumption of bounded Büchi automata we show that the test is co-NP-complete. Using bounded Büchi automata we are allowed to specify representatives of MSCs, thus the size of the specification can be smaller. More precisely, a bounded automaton might not accept an MSC language, however its MSC closure is still a regular language. Finally, without any buffer restriction the problem is shown to be undecidable. Our results extend the recent decidability results obtained for DFA and languages of finite MSCs in [6].

2 Definitions and Notations

In this section we define message sequence charts (MSC) and high-level message sequence charts (HMSC), based on the ITU standard MSC'96. Each MSC describes a scenario or an execution, where processes (instances) communicate with each other. An MSC scenario includes a description of the messages sent, messages received, the local events, and the ordering between them. Ordering of events is enforced by the instance ordering and the message ordering. In the visual description of MSCs, each instance is represented by a vertical line, which gives a total order on the events belonging to that instance. Messages are usually represented by horizontal or slanted arrows from the sending process to the receiving one, see Figure 1.

Throughout the paper we consider finite or infinite MSCs over a set of processes/instances $\mathcal{P} = \{P1, \ldots, Pn\}$ and we denote by $[n]$ the set of process indexes $\{1, \ldots, n\}$.

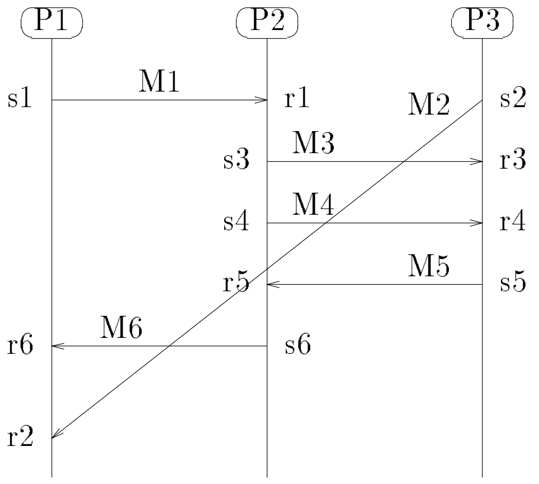


Fig. 1. Visual representation of an MSC.

Definition 1. *An MSC M over process set $\mathcal{P}$ is a tuple $\langle E, <, \mathcal{P}, \ell, t, m\rangle$:*

- *E is a set of events,*
- *$\ell : E \rightarrow [n]$ is a mapping that associates each event with a process (*location*),*
- *$t : E \rightarrow \{\mathrm{s}, \mathrm{r}, \mathrm{l}\}$ is a mapping that describes each event as send, receive or local (*type*),*
- *$m : t^{-1}(\mathrm{s}) \longrightarrow t^{-1}(\mathrm{r})$ is a bijection that pairs up send and receive events (*matching *function).*
- *$< \subseteq E \times E$ is an acyclic relation satisfying the two conditions below:*
  - *$\ell^{-1}(i)$ is totally ordered by $<$ for any instance Pi.*
  - *For all $e, f \in E$, $m(e) = f$ implies $e < f$.*

Note that we do not require that the set of events is finite. Therefore, an MSC is either finite or infinite.

A message (e, f) consists of a pair of matching send and receive events, i.e., $m(e) = f$. We assume that communication channels are fifo. Thus, we require that whenever the messages $m(e_1) = f_1$ and $m(e_2) = f_2$ are such that $\ell(e_1) = \ell(e_2)$ and $\ell(f_1) = \ell(f_2)$, we have $e_1 < e_2$ if and only if $f_1 < f_2$.

The relation $<$ is called the *visual order* of the MSC. It can be obtained from the syntactical representation of the chart. Since $<$ is required to be acyclic its reflexive-transitive closure is a partial order on E. A total order $\preceq$ on E is called a *linearization* of $<$, if $< \subseteq \preceq$. For simplifying our notations we will also write $\leq$ for the reflexive-transitive closure of $<$.

Event labels. For reasoning about linearizations of partial orders of MSCs we need event labels of the following kind. We label events by their type, location and the location of the matching event. That is, for every $e, f \in E$ such that $\ell(e) = i$, $\ell(f) = j$ and $m(e) = f$ we label e by $snd(i, j)$ and f by $rec(i, j)$. For example, both send events $s3, s4$ in Figure 1 will be labeled by $snd(2, 3)$.

The set of *send* and *receive labels*, respectively, is $\mathcal{L}_S = \{snd(i, j) \mid i, j \in [n]\}$ and $\mathcal{L}_R = \{rec(i, j) \mid i, j \in [n]\}$, respectively. Let $\mathcal{L} = \mathcal{L}_S \cup \mathcal{L}_R$ be the set of all *event labels*. If there is no risk of confusion we use the notions of *event* and *event label* interchangeably.

Remark 1. The results of this paper can be easily extended to MSCs with event and message names or local events. For this it suffices to extend the event labels by adding names. For simplicity we will consider in the following neither local events, nor message names. Let us also note that the extension can be done even if the fifo condition is restricted to messages with same name.

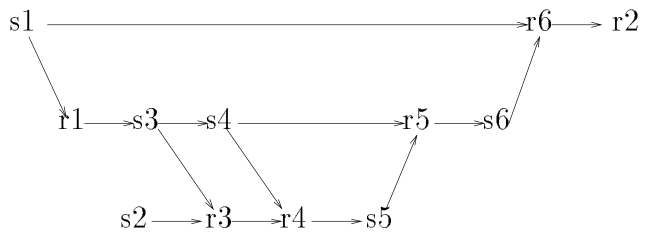


Fig. 2. The partial order between the events of the MSC in Figure 1.

The partial order between the send and receive events of Figure 1 is shown in Figure 2. In this figure we depicted the total order on each process and only the non-redundant message arcs.

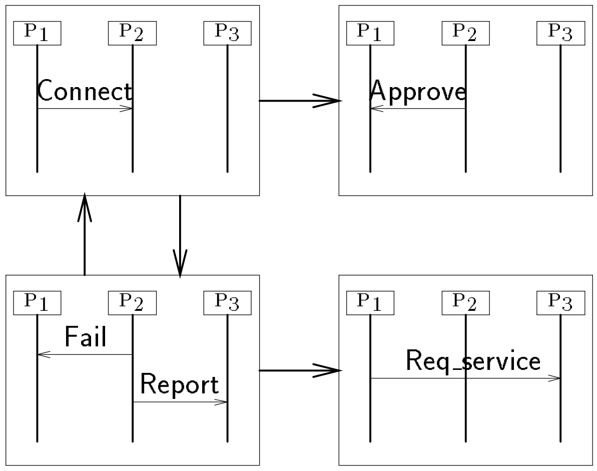
Definition 2. *The* concatenation *of two MSCs* $M_1 = \langle E_1, <_1, \mathcal{P}, \ell_1, t_1, m_1 \rangle$ *and* $M_2 = \langle E_2, <_2, \mathcal{P}, \ell_2, t_2, m_2 \rangle$ *over the same set of processes* $\mathcal{P}$ *is defined if for every process* Pi *either* $E_2 \cap \ell_2^{-1}(i) = \emptyset$ *or* $E_1 \cap \ell_1^{-1}(i)$ *is finite. When defined, it is the MSC* $M_1\, M_2 = \langle E_1 \cup E_2, <, \mathcal{P}, \ell_1 \cup \ell_2, t_1 \cup t_2, m_1 \cup m_2 \rangle$ *over the disjoint union of events* $E_1 \cup E_2$*, with the visual order given by:*

$$< \; = \; <_1 \cup <_2 \cup \{(e, f) \in E_1 \times E_2 \mid \ell_1(e) = \ell_2(f)\}\,.$$

That is, the events of M_1 precede the events of M_2 for each process, respectively. Note that there is no synchronization of the different processes when moving from one MSC to the next one (*weak sequencing*). Hence, it is possible that one process is still involved in some actions of M_1, while another process has advanced to an event of M_2. We can use the concatenation repeatedly, and define an infinite concatenation similarly.

Since a communication system usually includes many such scenarios, a high-level description is needed for combining them together. The standard description of MSC'96 consists of a directed graph called HMSC (high-level MSC), where each node contains a finite MSC (see the example below). Maximal paths in this graph (i.e., paths which cannot be extended) starting from a designated initial state, are called *executions* or *scenarios*.

Definition 3. *An HMSC N is a tuple $\langle S, \rightarrow, s_0, c, \mathcal{P}\rangle$ where $(S, \rightarrow, s_0)$ is a finite transition system with initial state $s_0 \in S$, with states labeled by finite MSCs over $\mathcal{P}$ and with pairwise disjoint sets. A state $s \in S$ is labeled by the MSC $c(s)$. An* execution *of N is a (finite or infinite) MSC $c(s_0)\, c(s_1)\, c(s_2) \ldots$ that corresponds to a maximal path $s_0 \rightarrow s_1 \rightarrow \cdots$ in $(S, \rightarrow, s_0)$.*



Given an HMSC N we denote the set of $\mathcal{L}$-labeled linearizations of the executions in N by $\mathrm{Lin}(N)$. More generally, for a set N of MSCs we let $\mathrm{Lin}(N)$ be the set of $\mathcal{L}$-labeled linearizations of the MSCs in N.

3 MSC Languages

In this section we consider the question whether a regular language is an MSC language, i.e., whether it is equivalent to $\mathrm{Lin}(N)$ for some finite or infinite set N of finite or infinite MSCs. Being an MSC language is a necessary condition for the HMSC equivalence. It turns out that testing whether a regular language is an MSC language is PSPACE-complete. Fortunately, a simple syntactic condition (diamond property, see Section 3.1) suffices for an automaton to accept an MSC language.

By regular language we mean languages given by (non-deterministic) Büchi automata over the set of event labels $\mathcal{L}$. Thus, a finite word is accepted whenever it reaches a final state, whereas an infinite word is accepted when it repeats infinitely often a final state. We consider languages L of finite or infinite sequences, i.e., $L \subseteq \mathcal{L}^\infty$, where $\mathcal{L}^\infty = \mathcal{L}^* \cup \mathcal{L}^\omega$.

We need some further notations. For $a \in \mathcal{L}$ and $v \in \mathcal{L}^\infty$ let $\#(v,a) \in \mathbb{N} \cup \{\infty\}$ denote the number of occurrences of symbol a in v. We call a word $w \in \mathcal{L}^\infty$ *well-formed*, if all receive events in w have matching sends. Formally, for every prefix v of w, we have $\#(v, \mathit{snd}(i,j)) \geq \#(v, \mathit{rec}(i,j))$ for all $i,j \in [n]$. A finite or infinite word $w \in \mathcal{L}^\infty$ is called *complete* if it is well-formed and $\#(w, \mathit{snd}(i,j)) = \#(w, \mathit{rec}(i,j))$ for all i,j. Let $a,b \in \mathcal{L}$ with $a = \mathit{snd}(i,j)$, $b = \mathit{rec}(i,j)$ for some i,j, then we write $(a,b) \in \mathcal{M}$.

Note that event labels provide all the necessary information for computing the MSC associated with a complete sequence, since communication is supposed to be fifo. Thus, we associate canonically the i-th event labeled $\mathit{snd}(i,j)$ with the i-th event labeled $\mathit{rec}(i,j)$ for all i.

Definition 4. *A language L of finite or infinite words over $\mathcal{L}$ is an MSC language if there is a set N of finite or infinite MSCs such that $L = \mathit{Lin}(N)$.*

Let $\ell(\mathit{snd}(i,j)) = i$ and $\ell(\mathit{rec}(i,j)) = j$. For $w \in \mathcal{L}^\infty$ and $i \in [n]$ we denote by $\pi_i(w)$ the projection of w on the subalphabet $\mathcal{L}_i = \{a \in \mathcal{L} \mid \ell(a) = i\}$ of events located on process Pi. The following alternative characterization of MSCs is well-known and easy to check:

Proposition 1. *A language $L \subseteq \mathcal{L}^\infty$ is an* MSC language *if and only if it consists only of complete words and it satisfies the following property:*
For any $u, v \in \mathcal{L}^\infty$ such that $\pi_i(u) = \pi_i(v)$ for all $i \in [n]$, we have $u \in L$ if and only if $v \in L$.

Checking the condition of Proposition 1 on a given Büchi automaton $\mathcal{A}$ over $\mathcal{L}$ can be done in polynomial space. We can test the inclusion $L_{\mathrm{MSC}}(\prod_{i\in[n]} \mathcal{A}_i) \subseteq L(\mathcal{A})$, where $L_{\mathrm{MSC}}(\mathcal{A}')$ denotes the set of complete words accepted by $\mathcal{A}'$ and $\mathcal{A}_i$ the projection of $\mathcal{A}$ on $\mathcal{L}_i$. We show below that this cannot be improved, that is, this problem is PSPACE-complete. The difficulty in checking whether a Büchi automaton accepts an MSC language stems from the partial order, and not from matching events, as shown in the next proposition.

Proposition 2. *Let $\mathcal{A}$ be a Büchi automaton over the alphabet $\mathcal{L}$. We can check in polynomial time whether $\mathcal{A}$ accepts only complete words.*

The algorithm in Proposition 2 is based on dynamic programming (e.g., Warshall's algorithm). For each pair (i,j) of instances we can view the automaton as a weighted graph where $\mathit{snd}(i,j)$ ($\mathit{rec}(i,j)$, resp.) is replaced by 1 (by -1, resp.), and every other label by 0. Then the question is whether every finite path has positive weight; moreover, any accepting path with only finitely many weights ± 1 must have a prefix of weight 0 including all weights ± 1.

The proof of the PSPACE lower bound is similar to the proof given in [8,12] for testing whether a regular language is closed under partial commutations.

Proposition 3. *Let $\mathcal{A}$ be a Büchi automaton over the alphabet $\mathcal{L}$. The question whether $L(\mathcal{A})$ is an MSC language is complete for PSPACE.*

3.1 Diamond automata

We consider throughout the paper Büchi automata such that all states are accessible from the initial state. The definition below presents a syntactic condition which will be shown to guarantee that a Büchi automaton where all states are final accepts an MSC language.

Definition 5 (Diamond property). *Let $\mathcal{A}$ be a Büchi automaton over $\mathcal{L}$. We say that $\mathcal{A}$ satisfies the* diamond property *if for all states p, q, r and all $a, b \in \mathcal{L}$ where $p \xrightarrow{a} q \xrightarrow{b} r$ and $\ell(a) \neq \ell(b)$ there is some state s of $\mathcal{A}$ such that $p \xrightarrow{b} s \xrightarrow{a} r$ whenever one of the following conditions is satisfied:*

1. *$(a, b) \notin \mathcal{M}$, or*
2. *$(a, b) \in \mathcal{M}$ and there exists a word $v \in \mathcal{L}^*$ with $q_0 \xrightarrow{v} p$ and $\#(v, a) > \#(v, b)$.*

Suppose that the Büchi automaton $\mathcal{A}$ satisfies the diamond property. Since every path in $\mathcal{A}$ is labeled by a well-formed sequence we have for all states p, q and symbols $(a, b) \in \mathcal{M}$ with $p \xrightarrow{ab} q$ two cases: Either $\#(v, a) > \#(v, b)$ for all $v \in \mathcal{L}^*$ with $q_0 \xrightarrow{v} p$ and there is a path $p \xrightarrow{ba} q$. Or $\#(v, a) = \#(v, b)$ for all $v \in \mathcal{L}^*$ with $q_0 \xrightarrow{v} p$ and there is no path labeled ba starting in p.

The diamond property means that two linearizations of the same finite MSC cannot be distinguished by the automaton $\mathcal{A}$, i.e., they reach the same set of states in $\mathcal{A}$. The diamond property is well-known from Mazurkiewicz trace theory and trace automata, see [3].

Proposition 4. *Let $\mathcal{A}$ be a Büchi automaton with the diamond property and where all states are final. Then the restriction of $L(\mathcal{A})$ to the set of complete words is an MSC language.*

Remark 2. Without the assumption on final states the diamond property is not sufficient for accepting an MSC language, as shown in Figure 3 (state 0 is initial and state 1 is final).

4 HMSC Languages

Some simple finite state communication protocols cannot be modelized as the set of executions of an HMSC. The example in Figure 4 appears e.g. in [4] and describes a scenario of the alternating bit protocol. Note that whenever we match the first unmatched send event, say e_1, we are forced to add g_1, which

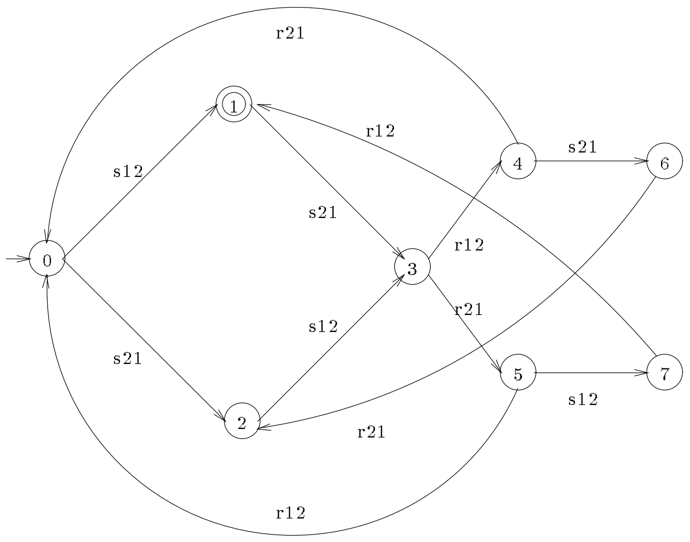


Fig. 3. Diamond property does not imply MSC language.

is not yet matched. Matching g_1 adds e_2 and so on. That is, the execution in Figure 4 has no finite complete prefix. In this section we propose algorithms for checking whether a specification given as a Büchi automaton can be turned into an equivalent HMSC specification.

Definition 6 (HMSC language). *A language $L \subseteq \mathcal{L}^\infty$ is called an HMSC language if some HMSC N exists such that $L = Lin(N)$.*

We want to test whether a Büchi automaton $\mathcal{A}$ accepts an HMSC language and if this is the case, compute an equivalent HMSC. As shown in [6] we just have to require that $L(\mathcal{A})$ is a finitely generated MSC language, see Theorem 1 below. Before stating the theorem we need some further notations. For a language $L \subseteq \mathcal{L}^\infty$ of complete words we define $[L]$ as the least MSC language $K \subseteq \mathcal{L}^\infty$ with $L \subseteq K$. We call $[L]$ the *MSC closure of* L. Let $w \in \mathcal{L}^\infty$. An *MSC decomposition* of w is a sequence of complete finite words $w_1, w_2, \ldots$ such that $w \in [w_1 w_2 \cdots]$.

The theorem below was shown in [6] (Thm. 4.5) for languages of finite MSCs given by deterministic finite automata.

Theorem 1. *Let $\mathcal{A}$ be a Büchi automaton accepting an MSC language. Then $L(\mathcal{A})$ is an HMSC language if and only if it is* finitely generated, *i.e., if there is a finite MSC language $G \subseteq \mathcal{L}^*$ such that $L(\mathcal{A}) \subseteq [G^\infty]$. Moreover, if such G exists then there is an HMSC N of size $O(|G| \cdot |\mathcal{A}|)$ with $L(\mathcal{A}) = Lin(N)$.*

4.1 Diamond automata and HMSC languages

The main result of [6] states that it is decidable whether a regular MSC language $L \in \mathcal{L}^*$ given by its minimal DFA is an HMSC language. Two interesting proper-

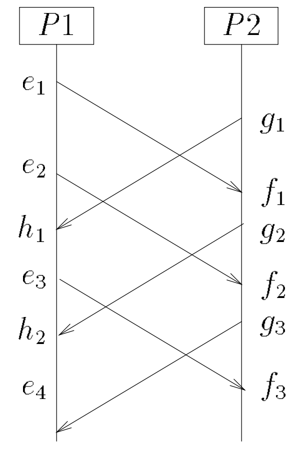


Fig. 4. A prefix of an infinite MSC execution that cannot be decomposed.

ties are satisfied by minimal automata of MSC languages, the diamond property and fixed buffer capacities. This means that we can associate with every state p and every pair $(a, b) \in \mathcal{M}$ a value $d(a, b)$ such that $|u|_a - |u|_b = d(a, b)$ for all words u labelling a path from the initial state to p. Furthermore, $d(a, b) < s$ where s is the size of $\mathcal{A}$. We denote a Büchi automaton satisfying this property an *automaton with fixed capacities*. Note that the state information of a finite state communication protocol usually determines the buffer sizes.

We extend and refine the result of [6] by showing that for any Büchi automaton $\mathcal{A}$ with the diamond property and fixed capacities we can test in polynomial time whether the accepted language is finitely generated. The next propositions describe the two cases where $L(\mathcal{A})$ is *not* finitely generated. Roughly speaking, the first case (Proposition 5) means that for a matching pair a, b of events we have a large number of events c between a and b. The second case (Proposition 6) corresponds to the situation in Figure 4.

As for MSCs there is a natural pairing between events (positions) in words. Let e be the i-th occurrence of $snd(i, j)$ and f the i-th occurrence of $rec(i, j)$ in v, then we say that *e matches f*. Let $uavbw$ be a complete word, $a \in \mathcal{L}_S$. If the event matching a belongs to the suffix bw, then we say that *a matches after b*. Note that with fixed capacities, the event matched by a in $uavbw$ is the same as in $u'avbw$, for all u, u' labelling paths that end in the same state. For any complete word $w \in \mathcal{L}^\infty$ we define a partial order $\leq$ between events of w in the same way we defined the visual order for MSCs. It is the partial order generated by $a < b$ in $w = taubv$ where either a matches b or $\ell(a) = \ell(b)$.

Proposition 5. *Let $\mathcal{A}$ be a Büchi automaton $\mathcal{A}$ with fixed capacities and satisfying the diamond property. Let s denote the number of states of $\mathcal{A}$. Assume that there is some $w \in L(\mathcal{A})$ of the form*

$$w = x\, a\, y\, a_1 b_1\, w_1 \cdots w_{m-1}\, a_m b_m\, w_m\, bz$$

where $a, a_i \in \mathcal{L}_S$, $b, b_i \in \mathcal{L}_R$, $x, y, w_i \in \mathcal{L}^*$, $z \in \mathcal{L}^\infty$, and $m \leq n$ satisfy the following conditions:

1. *a matches after b.*
2. *a_i matches b_i for all $1 \leq i \leq m$.*
3. *$\ell(a) = \ell(a_1)$, $\ell(b_1) = \ell(a_2), \ldots, \ell(b_{m-1}) = \ell(a_m)$ and $\ell(b_m) = \ell(b)$.*
4. *For some $1 \leq k \leq m$ we have $\sum_{e:\, \ell(e)=\ell(b_k)} \#(w_k, e) > s^2$.*

Then $L(\mathcal{A})$ is not finitely generated. Moreover, the conditions above can be checked in logarithmic space.

Proof. (Sketch.) By conditions 2,3 we have that $a < a_1 < b_1 < \cdots < a_m < b_m < b$ in w. Since a matches after b, it follows that the number of occurrences of b in $ya_1b_1w_1 \cdots a_mb_mw_m$ is at most s. By the last condition we can obtain some word w' from w by pumping a factor of w_k that contains no occurrence of b. Thus, a still matches after b in w'. For all new occurrences of e we must have $b_k < e < a_{k+1}$ in w'. Moreover, since $\mathcal{A}$ has fixed capacities, a_l still matches b_l in w' for all k. Since the earliest event matched by a is b and $a < b$, we have that a, b belong to the same factor in every MSC decomposition of w'. This means that arbitrary many events must belong to the same finite factor, contradiction.

For the complexity it suffices to guess a path and store $|x|_a - |x|_b \leq s$, checking against the number of b following a in $aya_1b_1w_1 \cdots a_mb_mw_m$. □

Proposition 6. *Let $\mathcal{A}$ be a Büchi automaton $\mathcal{A}$ with fixed capacities and satisfying the diamond property. Let s denote the number of states of $\mathcal{A}$. Assume that there is some $w \in L(\mathcal{A})$ of the form*

$$w = y_0 a_1\, y_1\, a_2\, x_1\, b_1\, y_2\, a_3\, x_2\, b_2 \cdots a_{t+1}\, x_t\, b_t\, z$$

where $a_i \in \mathcal{L}_S$, $b_i \in \mathcal{L}_R$, $x_i, y_i \in \mathcal{L}^*$, $z \in \mathcal{L}^\infty$ satisfy the following conditions:

1. *For all $1 \leq i \leq t$, a_i matches after b_i. Moreover, $\#(x_i, b_i) = 0$ and $(a_i, b_i) \in \mathcal{M}$ for all i.*
2. *For all $1 \leq i < t$, we have $a_i < a_{i+1} < b_i$ in w.*
3. *$t > sn$.*

Then $L(\mathcal{A})$ is not finitely generated. Moreover, the conditions above can be checked in logarithmic space.

Proof. (Sketch.) The first two conditions ensure that $a_i < a_{i+1} < b_i$ belong to the same factor in any MSC decomposition of w into MSCs. Thus, the events $a_1, \ldots, a_{t+1}, b_1, \ldots, b_t$ belong to the same MSC.

Since $t > sn$ we can assume that we have an accepting path in $\mathcal{A}$ which is labeled by $w = uxv$ and such that $x = a_ix_{i-1}b_{i-1}y_i \cdots a_{j-1}x_{j-2}b_{j-2}y_{j-1}$ is a loop where $a_{i-1} = a_{j-1}$, for some $i < j$. Consider some word $w' = uxx \cdots xv$ obtained from w by pumping x. We denote the k-th copy of x in w' as $x^{(k)}$. Assume that a_i, a_{j-1} in $x^{(k)}$ must belong to the same factor in any MSC decomposition. Note that the earliest event matched by a_{j-1} in $x^{(k)}$ is the event b_{i-1} in $x^{(k+1)}$.

Suppose that these two events belong to different factors in some MSC decomposition of w'. Then the factor containing b_{i-1} must precede the factor containing a_{j-1}. Since $a_i < b_{i-1}$ in $x^{(k+1)}$, the occurrences of a_i in $x^{(k)}$, $x^{(k+1)}$ must belong to a factor preceding a_{j-1}. This contradicts the fact that a_i and a_{j-1} in $x^{(k)}$ must belong to the same factor. By this argument we obtain that $a_1, \ldots, a_{i-1}$ and the occurrences of $a_i, \ldots, a_{j-1}$ in all $x^{(k)}$ belong to the same factor in any MSC decomposition of w'. Thus, arbitrary many events must belong to the same finite MSC factor, which yields a contradiction. For the complexity it suffices to guess a path and store the difference $\#(x_{i-1}y_i, a_i) - \#(x_{i-1}y_i, b_i) \leq s$ between a_i and a_{i+1}, for all i. □

Let $w = xaybz$, where $a, b \in \mathcal{L}$, $x, y, z \in \mathcal{L}^\infty$ and $a < b$ in w. A *$<$-path* from a to b is given by some factorization $y = y_0c_0y_1c_1 \cdots y_kc_ky_{k+1}$ of y such that $a < c_0 < \cdots < c_k < b$ in w.

Theorem 2. *The following question is complete for NSPACE*$(\log(n))$*, hence solvable in polynomial time:*

Let $\mathcal{A}$ be a Büchi automaton with fixed capacities and satisfying the diamond property. We ask whether $L(\mathcal{A})$ is finitely generated.

Proof. (Sketch.) For containment in NSPACE$(\log(n))$ we claim that any regular MSC language L that is not finitely generated satisfies the conditions of either Proposition 5 or 6. Informally, this means that either there is a pair of matching events a, b with a long $<$-path from a to b (Proposition 5). Or else matching events are always $<$-close, but there is some long sequence of send events $a_1 < a_2 < \cdots$ such that any MSC containing a_i must also contain a_{i+1} (Proposition 6). For the hardness we reduce from the question whether a directed graph is acyclic. □

Remark 3. From Propositions 5 and 6 it follows that if $L(\mathcal{A}) \subseteq [G^\infty]$ for some finite MSC language $G \subseteq \mathcal{L}^*$, then the length of any $<$-path in indecomposable G-words is bounded above by a polynomial in the number of states s and the number of processes n. Therefore all words in G have polynomial length, hence the size of G is at most exponential in s, n. This is also the upper bound for the size of an HMSC N with $\mathrm{Lin}(N) = L(\mathcal{A})$. The following example matches the exponential upper bound. Let $n = 3$ and let $a, \ldots, f$ denote the event labels $a = \mathit{snd}(1,2)$, $b = \mathit{rec}(1,2)$, $c = \mathit{snd}(1,3)$, $d = \mathit{rec}(1,3)$, $e = \mathit{snd}(3,2)$, $f = \mathit{rec}(3,2)$. The automaton below has 0 as initial state and $k+1$ as final state. It accepts a finite language of 2^{k-1} finite words. Every such word corresponds to a distinct MSC node in an equivalent HMSC.

$$\begin{tikzcd}
0 \arrow[r, "a"] & 1 \arrow[r, bend left, "cdef"] \arrow[r, bend right, "(cd)^2ef"'] & 2 \arrow[r, bend left, "cdef"] \arrow[r, bend right, "(cd)^2ef"'] & 3 & \cdots & (k-1) \arrow[r, bend left, "cdef"] \arrow[r, bend right, "(cd)^2ef"'] & k \arrow[r, "b"] & (k+1)
\end{tikzcd}$$

We describe now the algorithm for computing a finite set $G \subseteq \mathcal{L}^*$ of generators for $L(\mathcal{A})$, if $L(\mathcal{A})$ is finitely generated. Let ℓ be the upper bound on the length of elements of G. For each pair of states p, q of $\mathcal{A}$ of zero capacity we consider all paths of length at most ℓ from p to q. This can be done in an BFS way using partial-order reduction [1] (this avoids generating several linearizations of the same MSC factor). For each such path we check whether its label v is non-trivially decomposable into MSCs. If this is not the case, we add v to G. For checking whether v is decomposable we can e.g. directly adapt the following idea of [5]. With v we associate a directed graph G_v, where nodes correspond to events in v and edges (e, f) corresponding to $e < f$. Whenever e matches f we add the back edge (f, e). Then v is indecomposable if and only if G_v is strongly connected. We can compute G_v either using Tarjan's linear-time algorithm or on-line by an union-find algorithm keeping track of strongly connected components.

4.2 Bounded Büchi automata

The diamond property required for Büchi automata in Theorem 2 played a crucial role for the efficient test of finite generation. We show in this section that without this structural property, the question of finite generation for bounded specifications becomes co-NP-complete. We suppose in the following that we have given a Büchi automaton $\mathcal{A}$ accepting complete words. However, we drop the assumption that $L(\mathcal{A})$ is an MSC language. Instead, we require that $\mathcal{A}$ is bounded [2, 10]. This conditions ensures that the MSC closure of $L(\mathcal{A})$ is still regular, see Theorem 3 below. An automaton $\mathcal{A}$ is called *bounded*, if every word v labelling a loop in $\mathcal{A}$ satisfies the following condition. We associate a directed graph C_v with v. The nodes of C_v are process indices $i \in [n]$ and we have an edge from i to j if v contains the symbols $snd(i, j)$ and $rec(i, j)$. Then we require that C_v is strongly connected.

Theorem 3 ([2, 10]). *Let $\mathcal{A}$ be bounded and accepting only complete words. Then the MSC language $[L(\mathcal{A})]$ is regular and there is an automaton of size in $2^{poly(s,n)}$ accepting $[L(\mathcal{A})]$.*

The question we ask is whether there is any HMSC N such that $[L(\mathcal{A})] = \mathrm{Lin}(N)$. By Theorem 1 this problem reduces to checking whether $[L(\mathcal{A})]$ is finitely generated. Equivalently, we ask whether $L(\mathcal{A})$ is finitely generated, i.e., if there exists some finite MSC language $G \subseteq \mathcal{L}^*$ such that $L(\mathcal{A}) \subseteq [G^\infty]$. If such G exists then we obtain an HMSC N for $[L(\mathcal{A})]$ of size $O(|G| \cdot |\mathcal{B}|)$, with $L(\mathcal{B}) = [L(\mathcal{A})]$.

Proposition 7. *The following question is co-NP-hard:*
Let $\mathcal{A}$ be a bounded Büchi automaton over $\mathcal{L}$. We ask whether $L(\mathcal{A})$ is finitely generated.

Proof. (Sketch.) We give a reduction from 3-SAT. Let $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, with each C_i a clause with three literals over variables $x_1, \ldots, x_p$. We use m

processes $P_1, \ldots, P_m$ and event labels $a_i = \mathit{snd}(i, i+1)$, $b_i = \mathit{rec}(i, i+1)$, for all $1 \leq i \leq m$ with $m+1 = 1$. The automaton $\mathcal{A}$ has $p+1$ states $q_0, \ldots, q_p$ (for simplicity we label edges by words.) There are two edges from q_i to q_{i+1}, for $0 \leq i < p$. The first edge is labeled by the word $a_{i_1} \cdots a_{i_k}$, where $C_{i_1}, \ldots, C_{i_k}$ are all clauses where the variable x_{i+1} occurs positively. The second edge is labeled by the word $a_{j_1} \cdots a_{j_l}$, where $C_{j_1}, \ldots, C_{j_l}$ are all clauses where the variable x_{i+1} occurs negated. That is, any path from q_0 to q_p corresponds to a variable assignment. Moreover, the send labels occurring on the path to q_p correspond to clauses which are satisfied by the assignment. Finally, there is one self-loop around q_p, labeled by $a_1^3 b_1^3 \cdots a_m^3 b_m^3$. Clearly, $\mathcal{A}$ is bounded.

Consider a maximal path of $\mathcal{A}$ starting in q_0 and the associated MSC M. It is easy to see that if the first event of each process P_i, $i > 1$, is a send a_i and P_1 starts with more than three occurrences of a_1, then M is not decomposable as product of finite MSCs. This occurs when F is satisfied by the assignment. For the converse we show that the blocks of 3 consecutive b_i suffice for a decomposition in finitely many finite MSC factors. □

For the theorem below we show that any automaton that accepts a non-finitely generated language satisfies variants of Propositions 5 and 6, where the length of the counter-example is polynomial in the size of the automaton s and the number of instances n. For this bound we use that buffers are bounded by the size of the automaton.

Theorem 4. *The following question is complete for co-NP:*
Let $\mathcal{A}$ be a bounded Büchi automaton over $\mathcal{L}$, such that every loop is labeled by a complete word. We ask whether $L(\mathcal{A})$ is finitely generated.

Not surprisingly, we show that allowing unbounded message queues makes the problem of equivalence with an HMSC undecidable. The automaton model used in the proposition below are communicating finite state processes with fifo buffers (known also as message-passing automata). The proof actually shows that is undecidable whether $[L(\mathcal{A})]$ is an HMSC language, when $\mathcal{A}$ is not bounded.

Proposition 8. *It is undecidable whether a network of communicating finite state processes is equivalent to an HMSC.*

Proof. We use processes $P1, P2, P3, P4$ for a reduction from a PCP instance $(u_1, v_1), \ldots, (u_k, v_k)$ of pairs of words over some mutual alphabet A. We may assume that every solution $u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}$ is such that $i_n = k$. Moreover, for every sequence $i_1, \ldots, i_n$ with $i_n = k$ the word $v_{i_1} \cdots v_{i_n}$ is not a strict prefix of $u_{i_1} \cdots u_{i_n}$.

Messages from $P1$ to $P3$ and from $P3$ to $P2$ are named by the indices of the PCP pairs $1, \ldots, k$. Messages from $P1$ to $P2$ are named by the letters of A. Process $P1$ executes the following loop finitely often: it chooses $1 \leq i \leq k$ and sends first the message i (the pair index) to $P3$, then a sequence of messages to $P2$ corresponding to u_i. Process $P3$ loops: upon receiving i from $P1$, it retransmits it to $P2$. Process $P2$ verifies the solution guessed by $P1$, executing the following loop: upon receiving i from $P3$, process $P2$ is ready to receive a sequence of messages from $P1$ corresponding to v_i. If a mismatch occurs, $P2$ enters a special state s. In state s process $P2$ loops, alternating between sending a message to $P4$ and receiving a message from $P4$ (starting with a send). Meanwhile, $P2$ is ready to receive any message from $P1$ or $P3$. Assume now that $P2$ is not in state s. If $P2$ receives $i = k$ from $P3$ and all messages corresponding to v_k from $P1$ successfully, then it sends first a message to $P4$ and then starts alternating between sending a message to $P4$ and receiving a message from $P4$ (starting again with a send). Finally, $P4$ alternates between sending a message to $P2$ and receiving from $P2$ (starting with a send). We obtain a sequence of send and receive events between $P2$ and $P4$ which corresponds to the infinite, indecomposable MSC in Figure 4 if and only if the PCP instance has a solution. $\square$

References

1. R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Partial order reduction in symbolic state space exploration. In *Proc. of the 9th Int. Conf. on Computer Aided Verification (CAV'97)*, LNCS 1254, pages 340–351, 1997.
2. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. of the 10th Int. Conf. on Concurrency Theory (CONCUR'99)*, LNCS 1664, 1999.
3. V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
4. E. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. In *Proc. of the Tools and Algorithms for the Construction and Analysis of Systems, 7th International Workshop (TACAS'2001)*, 2001. To appear.
5. L. Hélouët and P. Le Maigat. Decomposition of Message Sequence Charts. In *Proc. of the 2nd Workshop on SDL and MSC (SAM'2000)*, pages 46–60, 2000.
6. J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. Thiagarajan. On message sequence graphs and finitely generated regular msc languages. In *Proc. of the 27th Int. Colloq. on Automata, Languages and Programming (ICALP'2000)*, LNCS 1853, pages 675–686, 2000.
7. V. Levin and D. Peled. Verification of message sequence charts via template matching. In *TAPSOFT (FASE)'97, Theory and Practice of Software Development*, LNCS 1214, pages 652–666, 1997.
8. A. Muscholl. *Über die Erkennbarkeit unendlicher Spuren.* Teubner, 1996.
9. A. Muscholl. *Decision and complexity issues on concurrent systems.* Habilitationsschrift, Universität Stuttgart, 1999.
10. A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proc. of the 24th Symp. on Math. Found. of Comp. Sci. (MFCS'99)*, LNCS 1672, pages 81–91, 1999.
11. A. Muscholl, D. Peled, and Z. Su. Deciding properties of message sequence charts. In *Proc. of the 1st Int. Conf. on Found. of Software Sci. and Comput. Structures (FoSSaCS'98)*, LNCS 1378, pages 226–242, 1998.
12. D. A. Peled, T. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages. *Theoretical Computer Science*, 195(2):183–203, 1998.