Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Theoretical Computer Science 331 (2005) 97–114

Theoretical Computer Science

www.elsevier.com/locate/tcs

# Realizability and verification of MSC graphs☆

Rajeev Alur[a,*], Kousha Etessami[b], Mihalis Yannakakis[c]

[a] *Department of Computer and Information Science, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104, USA*

[b] *Laboratory for Foundations of Computer Science, School of Informatics, The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland, UK*

[c] *Department of Computer Science, Columbia University, 1214 Amsterdam Avenue, New York, NY 10027, USA*

Received 30 November 2001; received in revised form 31 October 2003

## Abstract

Scenario-based specifications such as message sequence charts (MSC) offer an intuitive and visual way to describe design requirements. MSC-graphs allow convenient expression of multiple scenarios, and can be viewed as an early *model* of the system that can be subjected to a variety of analyses. Problems such as LTL model checking are undecidable for MSC-graphs in general, but are known to be decidable for the class of *bounded* MSC-graphs.

Our first set of results concerns checking *realizability* of bounded MSC-graphs. An MSC-graph is realizable if there is a distributed implementation that generates precisely the behaviors in the graph. There are two notions of realizability, *weak* and *safe*, depending on whether or not we require the implementation to be deadlock-free. It is known that for a finite set of MSCs, weak realizability is coNP-complete while safe realizability has a polynomial-time solution. We establish that for bounded MSC-graphs, weak realizability is, surprisingly, undecidable, while safe realizability is in EXPSPACE.

Our second set of results concerns verification of MSC-graphs. While checking properties of a graph $G$, besides verifying all the scenarios in the set $L(G)$ of MSCs specified by $G$, it is desirable to verify all the scenarios in the set $L^w(G)$—the *closure* of $G$, that contains the implied scenarios that any distributed implementation of $G$ must include. For checking whether a given MSC $M$ is a possible behavior, checking $M \in L(G)$ is NP-complete, but checking $M \in L^w(G)$ has a quadratic

☆A preliminary version of this paper appears in *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming* (ICALP'01), LNCS 2076, Springer, pp. 797–808, 2001. This research was partially supported by NSF Career award CCR97-34115, NSF award CCR99-70925, and Alfred P. Sloan Faculty Fellowship.

*Corresponding author.

*E-mail address:* alur@cis.upenn.edu (R. Alur).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.09.034

solution. For temporal logic specifications, considering the closure makes the verification problem harder: while checking LTL properties of $L(G)$ is PSPACE-complete for bounded graphs $G$, checking even simple "local" properties of $L^w(G)$ is undecidable.

© 2004 Elsevier B.V. All rights reserved.

*Keywords:* Formal verification; Software specification; Message sequence charts

## 1. Introduction

Message sequence charts (MSCs) are a commonly used visual notation for describing message exchanges between concurrent processes. They have become popular among software engineers for early requirements specification. Recently MSCs have been standardized by ITU [10], and incorporated in modern software engineering notations such as UML [7]. In the simplest form, an MSC depicts the desired exchange of messages, and corresponds to a single (partial-order) execution of the system. In recent years, a variety of features have been introduced so that a designer can specify multiple scenarios conveniently. In particular, *MSC-graphs* allow MSCs to be combined using operations such as choice, concatenation, and repetition. MSC-graphs can be viewed as an early model of the system that can be subjected to formal analysis. This has motivated the development of algorithms for a variety of analyses including detecting race conditions and timing conflicts [2], pattern matching [14], detecting non-local choice [4], and model checking [3], and tools such as uBET [9] and MESA [5].

An MSC-graph consists of a graph $G$ whose nodes are labeled by MSCs, and $G$ is viewed as defining the set $L(G)$ of all MSCs obtained by concatenating the MSCs that appear along any (directed) finite path from the designated start node of $G$. It is worth noting that the traditional high-level model for concurrent systems has been communicating state machines. Both communicating state machines and MSC-graphs can be viewed as specifying sets of behaviors, but the two offer dual views: the former is a parallel composition of sequential machines, while the latter is a sequential composition of concurrent executions. The complexity of a variety of verification questions in the communicating-state-machines model has been well understood: typically the problems are undecidable, and we must assume a bound on the sizes of message-buffers to obtain decidability results. Recent results indicate that verification problems about MSC-graphs are also undecidable in general as a process can send a potentially unbounded number of messages yet to be received [3,14]. The requirement for decidability, for problems such as LTL model checking, seems to be *boundedness*: in a bounded MSC-graph, in every cycle, for every pair of active processes $p$ and $q$, there is a sequence of communications from $p$ to $q$ and back, ensuring that all the active processes stay roughly synchronized, thereby bounding the number of pending messages [3,13]. The boundedness property of an MSC-graph can be checked in time exponential in the number of processes [3], and linear in the size of the MSC-graph. In this paper, we study a variety of analysis problems for bounded MSC-graphs.

The first analysis question studied in this paper concerns a form of consistency, called *realizability*, of specifications given as an MSC-graph. As observed in [1], a set of MSCs can potentially imply other, distinct, MSCs whose communication pattern must be exhibited by any concurrent system that realizes the given MSCs. An MSC-graph $G$ is said to be realizable if there exists a distributed implementation whose behaviors are precisely the ones specified by $G$. The precise definition of realizability depends on the underlying communication architecture for the distributed system [1]. In this paper we focus on realizability under a basic FIFO communication architecture. Unspecified, but implied, behaviors can be indicative of logical errors, and can be revealed by checking realizability. We prove that checking this form of realizability is, surprisingly, undecidable for bounded MSC-graphs by a reduction from the Post correspondence problem. Intuitively, this is because, while a bounded graph ensures boundedness of buffers in the scenarios specified in the graph, it does not ensure boundedness of buffers in its distributed implementation where different processes can follow different paths in the graph.

We study a second form of realizability, called *safe* realizability, where the distributed implementation must be *deadlock-free*. Safe realizability is a stronger notion of realizability, and corresponds to inferring partial global behaviors from local views of the specified MSCs. For a finite set of MSCs, checking weak realizability is coNP-complete, while checking safe realizability has a polynomial-time solution [1]. For bounded MSC-graphs, we show that checking safe realizability, unlike the weaker version, is decidable. We establish an upper bound of EXPSPACE. We show the problem is PSPACE-hard, but matching the lower and upper bounds remains an open problem.

For the purpose of verification of an MSC-graph $G$, due to the gap between an MSC-graph and its implementation, besides $L(G)$, we also consider $L^w(G)$, the *weak-closure* of $G$, containing all MSCs implied by MSCs in $G$, as a possible semantics. As we will see, a verification question can have different answers and different complexities depending upon this choice of semantics.

Our first verification problem concerns testing whether a given scenario $M$ is a possible behavior of a given MSC-graph $G$. This is relevant in identifying if a new scenario is already present in the existing specification, and also for detecting bugs if $M$ specifies an undesired scenario. We show that the problem of verifying whether $M \in L(G)$ is NP-complete in general, but can be solved in polynomial-time if the number of processes is bounded. We establish that testing whether $M$ is in the closure of $L(G)$ can be solved in quadratic time. This shows that it is easier to determine whether an MSC exists in the closure than in the originally given set, and furthermore, the questions about the implementation of $G$ can sometimes be verified without constructing it.

Finally, we consider the model checking problem, where the model is given by an MSC graph $G$ and the specification is given by automata or by temporal logic formulas. When the semantics of $G$ is $L(G)$, and the specification is given by an automaton accepting linearizations corresponding to “bad” behaviors, the problem is undecidable in general and PSPACE-complete for bounded graphs [3]. If the specification is given by “local” properties that do not distinguish between different linearizations of the same MSC, model checking can be solved in polynomial-time [15]. In this paper, we show that under the closure-semantics, the model checking questions become harder: for an acyclic graph the problem is coNP-complete, and for bounded graphs the problem is undecidable, even for simple linearization-invariant local specifications.

## 2. Specification languages

### 2.1. Message sequence charts

We start by recalling the definition of message sequence charts. Informally, a single MSC depicts the message exchanges in one communication scenario between entities of a concurrent system. For example, in Fig. 1 two MSCs are depicted giving two distinct communication scenarios in a client-server system where messages pass through a proxy. In the left scenario, the proxy simply relays the request message from the client to the server, while in the right scenario the proxy has a cached copy of the requested item, and hence responds to the client without involving the server.

Our formal definition of MSCs captures the essence of the ITU standard MSC'96, and is analogous to the definitions of labeled MSCs given in [1–3]. Let $\{P_1, \ldots, P_n\}$ be a set of processes, and $\Sigma$ be a message alphabet. We use the label $send(i, j, a)$ to denote the event "process $P_i$ sends the message $a$ to process $P_j$". Similarly, $receive(i, j, a)$ denotes the event "process $P_j$ receives the message $a$ from process $P_i$". Define the set $\Sigma^S = \{send(i, j, a) \mid 1 \leqslant i, j \leqslant n \;\&\; a \in \Sigma\}$ of *send labels*, the set $\Sigma^R = \{receive(i, j, a) \mid 1 \leqslant i, j \leqslant n \;\&\; a \in \Sigma\}$ of *receive labels*, and $\hat{\Sigma} = \Sigma^S \cup \Sigma^R$ as the set of *event labels*. A *$\Sigma$-labeled MSC $M$* over processes $\{P_1, \ldots, P_n\}$ is given by:

- a finite set $E$ of events which is partitioned into a set $S$ of "send" events and a set $R$ of "receive" events;
- a mapping $p$ that maps each event $e$ to a process $1 \leqslant p(e) \leqslant n$ on which it occurs;
- a bijective mapping $f : S \mapsto R$ between send and receive events, matching each send with its corresponding receive;
- a mapping $l : E \mapsto \hat{\Sigma}$ which labels each event such that $l(S) \subseteq \Sigma^S$ and $l(R) \subseteq \Sigma^R$, and furthermore for consistency of labels, for all $s \in S$, if $l(s) = send(i, j, a)$ then $p(s) = i$ and $l(f(s)) = receive(i, j, a)$ and $p(f(s)) = j$;
- for each $1 \leqslant i \leqslant n$, a total order $\leqslant_i$ on the events of process $P_i$, that is, on the elements of $p^{-1}(i)$, such that the transitive closure of the relation

$$\leqslant \;\doteq\; \bigcup_{1 \leqslant i \leqslant n} \leqslant_i \;\cup\; \{(s, f(s)) \mid s \in S\}$$

is a partial order on $E$.

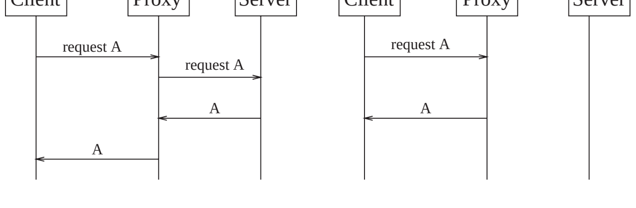

Fig. 1. Two simple MSCs depict client-server scenarios through a proxy.

We require all our MSCs to satisfy an additional FIFO condition:

- there is no reversal of the order in which two messages sent by some process $P_i$ are received by another process $P_j$, that is, for send events $s_1, s_2 \in S$, if $p(s_1) = p(s_2) = i$, and $s_1 \leqslant_i s_2$, then $f(s_1) \leqslant_j f(s_2)$, where $j = p(f(s_1))$.

If the underlying architecture is not FIFO, then a weaker *non-degeneracy* condition can be used. Non-degeneracy condition disallows reversals between a pair of *identical* messages between a given pair of processes [1]. The results of this paper are developed using the FIFO condition, but we will indicate when they also hold with the non-degeneracy condition.

Observe that the information in MSCs can be captured by any word over $\hat{\Sigma}$ that corresponds to the sequence of labels of any linearization that is consistent with the partial order $\leqslant$. Furthermore, any word over $\hat{\Sigma}$ in which the *send* and *receive* events can be matched, uniquely defines an MSC. Let us be more precise. A word $w = w_1 \cdots w_{|E|}$ over the alphabet $\hat{\Sigma}$ is a *linearization* of an MSC $M$ iff there exists a total order $e_1 \cdots e_{|E|}$ of the events in $E$ such that whenever $e_i \leqslant e_j$, we have $i \leqslant j$, and for $1 \leqslant i \leqslant |E|$, $w_i = l(e_i)$. Let $w$ be a word over $\hat{\Sigma}$, and consider processes $i$ and $j$. We define the projections $w \Uparrow \mathit{send}(i,j)$ and $w \Uparrow \mathit{receive}(i,j)$ as follows. If $w$ is the empty word, then $w \Uparrow \mathit{send}(i,j)$ and $w \Uparrow \mathit{receive}(i,j)$ equal the empty word. Suppose $w = xv$, for $x \in \hat{\Sigma}$. If $x = \mathit{send}(i,j,a)$ then $w \Uparrow \mathit{send}(i,j) = a(v \Uparrow \mathit{send}(i,j))$ else $w \Uparrow \mathit{send}(i,j) = v \Uparrow \mathit{send}(i,j)$. If $x = \mathit{receive}(i,j,a)$ then $w \Uparrow \mathit{receive}(i,j) = a(v \Uparrow \mathit{receive}(i,j))$ else $w \Uparrow \mathit{receive}(i,j) = v \Uparrow \mathit{receive}(i,j)$. Now, a word $w$ is *well-formed* if for every prefix $v$ of $w$, for every pair of processes $i$ and $j$, $v \Uparrow \mathit{receive}(i,j)$ is a prefix of $v \Uparrow \mathit{send}(i,j)$. A word $w$ is *complete* if for every pair of processes $i$ and $j$, $w \Uparrow \mathit{send}(i,j) = w \Uparrow \mathit{receive}(i,j)$. A word $w$ over $\hat{\Sigma}$ is a linearization of an MSC iff it is well-formed and complete [1].

### 2.2. MSC graphs

A natural way to structure multiple scenarios is to employ graphs whose nodes are MSCs. Formally, an MSC-graph $G$ consists of a set $V$ of vertices, a binary relation $\rightarrow$ over $V$, an initial vertex $v^{\mathrm{I}}$, a set of terminal vertices $V^{\mathrm{T}}$, and a labeling function $\mu$ that maps each vertex $v$ to an MSC. The paths that start at the initial vertex and end at a terminal vertex represent (finite) *accepting paths* of $G$, i.e., the finite executions of the system modeled by the MSC-graph. To formally associate a set of MSCs with the MSC-graph $G$, we first have to define a concatenation operation on MSCs. Concatenation $M \cdot M'$ corresponds to a natural process-by-process pasting of the two MSCs $M$ and $M'$ together (see [3] for a formal definition). Then, we can associate an MSC with each path by concatenating MSCs corresponding to individual vertices. The (finite) language $L(G)$ of the graph is then all MSCs of the form $\mu(v_0) \cdot \mu(v_1) \cdots \mu(v_n)$, where $v_0 v_1 \ldots v_n$ is an accepting path in $G$. Since MSCs are uniquely characterized by their linearizations, we will also use $L(G)$ to denote the set of all linearizations of the MSCs in it.

In general, the set $L(G)$ is not regular. The problem arises, for instance, when there is a cycle in the graph such that some process sends a message at some vertex in the cycle, but does not receive any message at any vertex in the cycle. For example, consider the MSC-graph with a single node with a self-loop, where the MSC associated with the node consists of a single message edge. The language of this MSC graph is non-regular, because it consists of strings of send's and receive's which are isomorphic to "properly parenthesized" expressions over the alphabet $\{(,)\}$, a language known not to be a regular language. The class of bounded MSCs avoids this problem. Given an MSC-graph $G$ and a subset $U$ of its vertices, define the communication graph $H_U$ of $U$ as follows: the set of vertices of $H_U$ is the set $P$ of all the processes, and there is an arc from process $p$ to process $q$ if $p$ sends a message to $q$ in the MSC $\mu(v)$ for some $v \in U$. For a set $U$ of vertices, we denote by $P_U$ the set of processes that send or receive a message in the MSC of some vertex in $U$, and call them the active processes of the set $U$. We call an MSC-graph *bounded* if for every cycle $\rho$ of $G$, the subgraph of the communication graph $H_\rho$ induced by the set $P_\rho$ of active processes of the cycle is strongly connected. In other words, communication graph $H_\rho$ on all the processes consists of one nontrivial strongly connected component and isolated nodes corresponding to processes that are inactive throughout the cycle. In [3], it is shown that if $G$ is bounded, the set of linearizations of all the MSCs in $L(G)$ is regular, and can be generated by a nondeterministic automaton whose size is exponential in the size of $G$. The converse of the question, namely, characterizing regular languages using MSC graphs, is studied in [8].

## 2.3. Concurrent automata

Our concurrency model is based on the standard buffered message-passing model of communication. There are several choices to be made with regard to the particular communication architecture of concurrent processes, such as synchrony/asynchrony and the queuing disciplines on the buffers. We fix our architecture to a standard asynchronous setting, with FIFO message buffers between all pairs of processes. We now formally define our automata $A_i$, and their (asynchronous) product $\Pi_{i=1}^n A_i$, which captures their joint behavior.

As in the previous section, let $\Sigma$ be the message alphabet. Let $\hat{\Sigma}_i$ be the set of labels of events belonging to process $P_i$, namely, the messages of the form *send*$(i, j, a)$ and *receive*$(j, i, a)$. The behavior of process $P_i$ is specified by an automaton $A_i$ over the alphabet $\hat{\Sigma}_i$ with the following components: (1) a set $Q_i$ of states, (2) a transition relation $\delta_i \subseteq Q_i \times \hat{\Sigma}_i \times Q_i$, (3) an initial state $q_i^0 \in Q_i$, and (4) a set $F_i \subseteq Q_i$ of accepting states.

To define the joint behavior of the set of automata $A_i$, we need to describe the message buffers. For each ordered pair $(i, j)$ of process indices, we have two message buffers $B^s_{i,j}$ and $B^r_{i,j}$. The first buffer, $B^s_{i,j}$, is a "pending" buffer which stores the messages that have been sent by $P_i$ but are still "in transit" and not yet accessible by $P_j$. The second buffer $B^r_{i,j}$ contains those messages that have already reached $P_j$, but are not yet accessed and removed from the buffer by $P_j$. All the buffers are words over the message alphabet $\Sigma$. We define the asynchronous product automaton $A = \Pi_{i=1}^n A_i$ over the alphabet $\hat{\Sigma}$, given by:

*States*. A state $q$ of $A$ consists of the (local) states $q_i$ of component processes $A_i$, along with the contents of the buffers $B^s_{i,j}$ and $B^r_{i,j}$.

*Initial state*. The initial state $q_0$ of $A$ is given by having the component for each process $i$ be in the start state $q_i^0$, and by having every buffer be empty.

*Transitions*. In the transition relation $\delta \subseteq Q \times (\hat{\Sigma} \cup \{\tau\}) \times Q$, the $\tau$-transitions model the transfer of messages from the sender to the receiver. The transitions are defined as

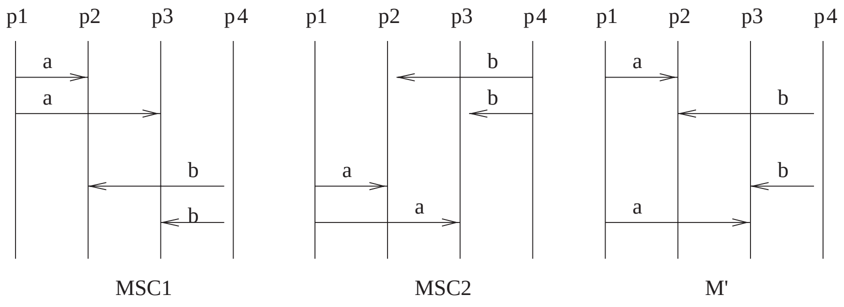

Fig. 2. Weak inference.

follows:

(1) For an event $x \in \hat{\Sigma}_i$, $(q, x, q') \in \delta$ iff (a) the local states of processes $k \neq i$ are identical in $q$ and $q'$, (b) the local state of process $i$ is $q_i$ in $q$ and $q'_i$ in $q'$ such that $(q_i, x, q'_i) \in \delta_i$, (c) if $x = receive(j, i, a)$ then the buffer $B^r_{j,i}$ in state $q$ contains the message $a$ in the front, and the corresponding buffer in state $q'$ is obtained by deleting $a$, (d) if $x =$ *send*$(i, j, a)$, the buffer $B^s_{i,j}$ in state $q'$ is obtained by appending the message $a$ to the corresponding buffer in state $q$, and (e) all other buffers are identical in states $q$ and $q'$.

(2) There is a $\tau$-labeled transition from state $q$ to $q'$, iff states $q$ and $q'$ are identical except that for one pair $(i, j)$, the buffer $B^s_{i,j}$ in state $q'$ is obtained from the corresponding buffer in state $q$ by deleting the first message $a$, and the buffer $B^r_{i,j}$ in state $q'$ is obtained from that in $q$ by adding that message $a$ at its end.

*Accepting states*. A state $q$ of $A$ is accepting if for all processes $i$, the local state $q_i$ of process $i$ in $q$ is accepting, and all the buffers in $q$ are empty.

We associate with $A = \Pi_i A_i$ the language of possible executions of $A$, denoted $L(A)$, which consists of all those words in $\hat{\Sigma}^*$ leading $A$ from start state $q_0$ to an accepting state, where $\tau$-transitions are viewed as $\varepsilon$-transitions in the usual automata-theoretic sense. For any set of concurrent automata $A_i$, the language $L(\Pi_i A_i)$ of the product of the automata contains only complete and well-formed words. Furthermore, for a given MSC $M$, the language $L(\Pi_i A_i)$ either contains all linearizations of $M$ or it contains none.

## 3. Realizability

### 3.1. Weak realizability

Consider the two MSCs MSC1 and MSC2 shown in Fig. 2. Any distributed implementation that exhibits these two behaviors must also exhibit the behavior depicted by $M'$. This is because, as far as each process can locally tell, the scenario is proceeding according to one of the two given scenarios. Consequently, we say that the set of MSCs containing MSC1 and MSC2 (weakly) implies $M'$ [1].

Formally, given a set $L$ of MSCs (or equivalently, their linearizations), and another MSC $M$, we say that $L$ *weakly implies* $M$, if for any sequence of automata $\langle A_i \mid 1 \leqslant i \leqslant n \rangle$, if every MSC in $L$ is in $L(\Pi_i A_i)$ then so is $M$ in $L(\Pi_i A_i)$. The *weak closure* $L^w$ of a set $L$ of MSCs contains all the MSCs $L$ weakly implies, and the set $L$ is weakly realizable iff $L = L^w$. The notions defined above naturally extend to MSC-graphs. The MSC-graph $G$ is said to be *weakly realizable* if the set $L(G)$ of MSCs is. Thus, a weakly realizable graph already contains all the implied scenarios.

For computational purposes, an alternative characterization of the weak realizability is helpful. For an MSC $M$ and a process $P_i$, let $M|_i$ denote the sequence of events belonging to the process $P_i$ in $M$. Then, a set $L$ of MSCs weakly implies an MSC $M$ iff for all $1 \leqslant i \leqslant n$, there exists an MSC $M_i \in L$ such that $M|_i = M_i|_i$ [1]. In other words, for every process $P_i$, the events occurring on $P_i$ in MSC $M$ are consistent with the events occurring on $P_i$ in some MSC known to be in the language $L$, then $M$ is implied, and $M$ must be in $L$ for $L$ to be closed. Intuitively, a closed language $L$ can be constructed from the projections of the MSCs in $L$ onto individual processes. For a finite set of MSCs, checking weak realizability is coNP-complete [1]. We show that checking weak realizability is undecidable for bounded graphs.

**Theorem 1.** *Given a bounded MSC graph $G$, checking if $G$ is weakly realizable is undecidable.*

**Proof.** The proof is a reduction from the *post correspondence problem* (PCP). The PCP is as follows: given a collection of pairs $\langle (v_1, w_1), (v_2, w_2), \ldots, (v_r, w_r) \rangle$, where $v_i, w_i \in \Sigma^*$, for some fixed finite alphabet $\Sigma$, with designated initial pair $(v_1, w_1)$, determine whether there is a sequence of indices $i_2, \ldots, i_m$, such that

$$v_1 v_{i_2} \ldots v_{i_m} = w_1 w_{i_2} \ldots w_{i_m}. \tag{1}$$

By examining the standard proof of undecidability for the PCP from the Turing machine halting problem, one can see that the constructed PCP instance has the property that if there is a solution then there is one where the one string is always a prefix of the other. In particular, the following version, call it OneSidedPCP, remains undecidable: determine whether there is a sequence of indices $i_2 \ldots i_m$, such that equality 1 holds, and furthermore, for all $j \leqslant m$, the string $w_1 w_{i_2} \ldots w_{i_j}$ is a prefix of the string $v_1 v_{i_2} \ldots v_{i_j}$ (that is, the right string never overtakes the left one). We will reformulate OneSidedPCP slightly further to suit our purposes. Let *Relaxed* PCP (RPCP) be the following problem: given $\{(v_1, w_1), (v_2, w_2), \ldots, (v_r, w_r)\}$, determine whether there are indices $i_1, \ldots, i_m$ such that $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$, where $x_{i_j}, y_{i_j} \in \{v_{i_j}, w_{i_j}\}$, for some index $i_l$ $x_{i_l} \neq y_{i_l}$, and for all $j \leqslant m$, $y_{i_1} \ldots y_{i_j}$ is a prefix of $x_{i_1} \ldots x_{i_j}$. □

We now prove that RPCP is undecidable.

**Lemma 2.** *RPCP is undecidable.*

**Proof.** Given an instance $\Delta = \langle (v_1, w_1), (v_2, w_2), \ldots, (v_r, w_r) \rangle$ of the OneSidedPCP problem, we will reduce it to RPCP as follows: introduce three new symbols: #, \$, and $\beta$ to the alphabet $\Sigma$, and call the new alphabet $\Sigma'$. We first make the following transformation

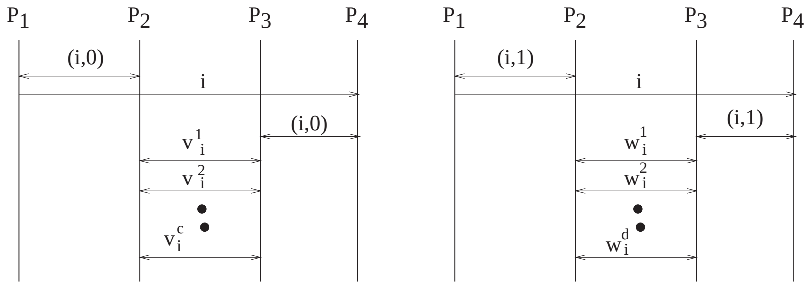

Fig. 3. MSCs $M_i^0$ and $M_i^1$.

on the words $v_i$ and $w_i$. For $a \in \Sigma$, let $h_v(a) = a\beta$, and let $h_w(a) = \beta a$. Extend $h_v$ and $h_w$ in the obvious way to a homomorphism from $\Sigma^*$ to $\Sigma'^*$. Let $v'_i = h_v(v_i)$ and let $w'_i = h_w(w_i)$.

We map an instance $\Delta$ of OneSidedPCP, to the following instance of RPCP:

$$\Delta' = \{(\#\beta v'_1, \#w'_1), (v'_1, w'_1), \ldots, (v'_r, w'_r), (\$, \beta\$)\}.$$

*Claim*: $\Delta \in OneSided$PCP if and only if $\Delta' \in$ RPCP.

To see the "only if" direction, note that if $v_1 v_{i_2} \ldots v_{i_m} = w_1 w_{i_2} \ldots w_{i_m}$ then $\#\beta v'_1 v'_{i_2} \ldots v'_{i_m}\$$ equals $\#w'_1 w'_{i_2} \ldots w'_{i_m}\beta\$$, and for $j < m$, $|\#\beta v'_1 v'_{i_2} \ldots v'_{i_j}|$ exceeds $|\#w'_1 w'_{i_2} \ldots w'_{i_j}|$.

To see the "if" direction, suppose $(i_1, \ldots, i_m)$ are a sequence of indices for a solution to $\Delta'$. Since there must be some index $i_j$ for which the chosen $x_{i_j}, y_{i_j}$ differ, let $i_j$ be the first such index in the sequence. Then, w.l.o.g., $x_{i_j} = \#\beta v'_1$ and $y_{i_j} = \#w'_1$, because for all other pairs one of them begins with $\beta$ while the other doesn't. Note that since the "v" string thus far ends with $\beta$, while the "w" string does not, in the next choice of pairs, we must choose $v'_{i_{j+1}}$ and $w'_{i_{j+1}}$ to append to the "$v$" and "w" strings, respectively. Proceeding in this way, we must end our string with the pair \$ and $\beta$\$, respectively. Now, if we eliminate the initial symbol #, the final symbol \$, and all the intermediate $\beta$ symbols from our solution for $\Delta'$, beginning at the first distinct pair $x_{i_j}, y_{i_j}$, we obtain a solution for $\Delta$. That establishes the claim. □

Now we reduce RPCP to weak realizability. Given a finite set $L$ of MSCs, let $L^*$ denote the MSC graph that consists of the complete graph with $|L|$ vertices one per MSC in the set $L$, dummy initial and terminal vertices $v_I$, $v_T$ with empty MSCs, and edges from $v_I$ to all vertices of $L$ and from those to $v_T$. Thus, an MSC of this graph is simply a concatenation of MSCs from the set $L$. In the sequel, we say that a process $p$ synchronously sends a message $m$ to process $q$, if $p$ sends $m$ to $q$ immediately followed by $q$ sending the message $m$ back to $p$. In figures, such messages will be depicted by double arrows.

Given an instance $\Delta = \{(v_1, w_1), \ldots, (v_m, w_m)\}$ of RPCP, we build a set $L$ of MSCs over 4 processes as follows. For a string $u$, let $u^l$ denote the $l$'th character of the string. For each pair $(v_i, w_i)$ we build two MSCs $M_i^0$ and $M_i^1$, which are depicted in Fig. 3. Thus in $M_i^0$, process 1 sends synchronously $(i, 0)$ to process 2 then sends the index $i$ to process 4, and then process 4 sends synchronously $(i, 0)$ to process 3. After that, process 2 synchronously sends the sequence of characters of $v_i$ to process 3 (note we assume $c$ is the length of $v_i$ and $d$ the length of $w_i$ in the figure), $M_i^1$ is similar. Observe that the communication graph of each of these MSCs is strongly connected and involves all the processes, and hence, the MSC graph $L^*$ is bounded.

*Claim* 1: *$\Delta \in RPCP$ iff $L^*$ is not weakly realizable.*

**Proof.** For the "only if" direction, suppose $R = (i_1, a_1, b_1, i_2, a_2, b_2, \ldots, i_m, a_m, b_m)$ are the indices for a solution to $\Delta$, and the bits $a_j$ and $b_j$ indicate which string ($v_{i_j}$ or $w_{i_j}$) is chosen to go into the two (left and right) long strings.

Consider the new MSCs $M$ and $M'$ obtained from the sequences $M = M_{i_1}^{a_1} \ldots M_{i_m}^{a_m}$ and the sequence $M' = M_{i_1}^{b_1} \ldots M_{i_m}^{b_m}$. Executions of both of these (sequences of) MSCs must exist in any realization of $L^*$. We then look at the projections $M|_1$, $M|_2$, $M|_3$, and $M|_4$ of $M$, and $M'|_1$, $M'|_2$ , $M'|_3$ and $M'|_4$ of $M'$ onto the 4 processes. Now consider an MSC $M''$ formed from $M'|_1$, $M'|_2$, $M|_3$, and $M|_4$. The claim is that the combined MSC $M''$ is weakly implied by $L^*$. By definition, the only thing to establish is that $M''$ is indeed an MSC, in the sense that it is acyclic, well-formed and complete. The only new situation in terms of communication in $M''$ is the communication between $P_1$ and $P_4$, and between $P_2$ and $P_3$. But the communication between $P_1$ and $P_4$ is consistent in $M'|_1$ and $M|_4$ (i.e. the sequence of messages sent from $P_1$ to $P_4$ in $M'|_1$ is equal to the sequence of messages received in $M|_4$), and the communication between $P_2$ and $P_3$ is consistent in $M'|_2$ and $M|_3$ because $R$ is a solution to the RPCP. Furthermore, the acyclicity of $M''$ follows from the property of the solution that the string formed by the first $j$ words on processes 1 and 2 is always a prefix of the string formed by the first $j$ words on processes 3 and 4. Consequently each message from $P_1$ to $P_4$ is sent before it needs to be received. But note that $M''$ cannot itself be in $L^*$ because there must be some index $i_j$ where $a_j \neq b_j$, and no MSC exists in $L$ where, after process 1 announces the index, what process 2 sends is not identical to what process 3 receives.

Now, for the "if" direction, suppose there is some MSC $M^@$ which exists in any realization of $L^*$, but is not in $L^*$ itself. We want to derive a solution to $\Delta$ from $M^@$. First, it is clear that the projection $M^@|_1$ must consist of a sequence of pairs of messages (the first of each pair acknowledged), sent from process 1 to process 2 and 4, respectively, with messages $(i, b)$ and $i$, respectively. Likewise, it is clear that, in order for process 2 to receive those messages, $M^@|_2$ must consist of a sequence of receipts of $(i, b)$ pairs, and after each $(i, b)$, either $v_i$ or $w_i$ is sent to process 3, based on whether $b = 0$ or $b = 1$, before the next index pair is received. Likewise $M^@|_4$ consists of a sequence of receipt of an index $i$ from process 1 followed by sending of $(i, 0)$ or $(i, 1)$ to process 3, and $M^@|_3$ consist of a sequence of receipt of $(i, 0)$ or $(i, 1)$ followed by receipt $v_i$ or $w_i$, respectively. Now, since $M^@$ is not in $L^*$, for some index $i$ the choice of $v_i$ or $w_i$ must differ on process 2 and process 3. (Note, we are assuming that the buffers between processes are FIFO.) Furthermore, because of the precedences, the prefix formed by the first $j$ words on process 2 must precede the $(j + 1)$th message from process 1 to process 4, which in turn precedes the $(j + 1)$th message from 4 to 3, and hence the $(j + 1)$th word on process 3. That is, the string formed by the first $j$ words on process 2 is a prefix of the string formed by the first $j$ words on process 3. Therefore, we can readily build a solution for $\Delta$ from $M^@$. $\square$

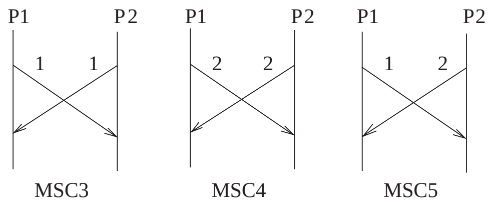

Fig. 4. Safe realizability.

## 3.2. *Safe realizability*

As a motivation for safe realizability, consider the MSCs in Fig. 4. In MSC$_3$, both processes send each other the value 1, while in MSC$_4$, both processes send each other the value 2, and thus, they agree in both cases. From these two, we should be able to infer a partial scenario, depicted in MSC$_5$, in which the two processes start by sending each other conflicting values, and the scenario is then completed in some way. However, the set containing only MSC$_3$ and MSC$_4$ is weakly realizable. A closer examination reveals that the distributed implementation of these two scenarios can potentially deadlock when one process decides to send the message 1 while the other decides to send the message 2. We need a stronger version of implication closure.

To define this formally, consider a set $A_i$ of concurrent automata and the product $A = \Pi_i A_i$. A state $q$ of the product $A$ is said to be a *deadlock* state if no accepting state of $A$ is reachable from $q$. For instance, a rejecting state in which all processes are waiting to receive messages which do not exist in the buffers will be a deadlock state. The product $A$ is said to be *deadlock-free* if no state reachable from its initial state is a deadlock state. A set $L$ of MSCs is said to be *safely realizable* if $L = L(\Pi A_i)$ for some $\langle A_i | 1 \leqslant i \leqslant n \rangle$ such that $\Pi A_i$ is deadlock-free. [1]

There is an equivalent characterization of safe realizability as follows. Let *pref*$(L)$ denote the set of prefixes of the MSCs or words in $L$. Then, a set $L$ of MSCs is safely realizable iff it satisfies the following two closure conditions:

(1) for a well-formed word $w$ (i.e. a partial MSC), if for all $1 \leqslant i \leqslant n$, there exists a word $v_i \in$ *pref*$(L)$ such that $w|_i = v_i|_i$, then $w$ is in *pref*$(L)$;
(2) for a well-formed and complete word $w$ (i.e. an MSC) in *pref*$(L)$, if for all $1 \leqslant i \leqslant n$, there exists a word $v_i \in L$ such that $w|_i = v_i|_i$, then $w$ is in $L$.

The first closure condition says that the set of partial MSCs (i.e. prefixes of $L$) can be constructed from the projections of the MSCs in $L$ onto individual processes. The second condition is similar to the closure condition for weak realizability, but allows us to focus attention only on complete MSCs that are themselves prefixes of MSCs already in $L$. The second condition is not implied by the first, as pointed out in [11]. For the proof that these two conditions capture safe realizability, please consult the full version of [1].

[1] Recall that we identify MSCs with their linearizations, and thus, a set of MSCs with the set of all linearizations of all the MSCs in that set.

The MSC-graph $G$ is said to be safely realizable if the set $L(G)$ of MSCs is. For a finite set of MSCs, we known that weak realizability is coNP-complete while safe realizability has a polynomial-time solution [1]. For bounded graphs, even though weak realizability is undecidable, checking safe realizability is decidable. In bounded graphs, if we consider the behaviors corresponding to the paths in the graphs, a process cannot be far ahead of its communication partner, thus keeping the buffer size bounded. While checking safe realizability, when we consider the possible interactions among local behaviors (i.e. projections) of different processes, if the communication buffer between any pair of processes exceeds this bound, we can immediately flag an error. In contrast, while checking weak realizability, even when the buffer size exceeds the bound, we need to check if there is a “complete” MSC that can extend this partial behavior. We establish an EXPSPACE upper bound, as well as PSPACE-hardness, for checking safe realizability. Note that both the bounds also hold if we use alternate communication architecture by relaxing the FIFO requirement on buffers.

**Theorem 3.** *Checking safe realizability of a bounded MSC-graph is in* EXPSPACE.

**Proof.** Since $G$ is bounded, we know that $L(G)$ is definable by an exponential sized automaton $A$ each of whose states can be encoded in polynomial space [3]. Likewise, we can build a concurrent product $A' = \Pi_i A_i$, where each $A_i$ is the local automaton formed by the projection onto process $i$ of $G$, and then determinized and minimized. If $L(G)$ is safely realizable, then we know that $A'$ is such a realization [1]. Moreover, since $G$ is bounded, there is a polynomial bound (actually, linear in the number of vertices of $G$) that we can place on the lengths of queues in $A'$ such that if ever the queue length is exceeded we will know that the partial MSC which exceeded the bound is not a prefix of an MSC in $L(G)$. Thus, we first check to see whether there is an execution of $A'$ in which the buffer bound is exceeded. This can be done in PSPACE by guessing a bad path. If there is such an execution, we halt and report that $L(G)$ is not realizable. Thus, we assume that $A'$ enforces the polynomial bound on the buffers.

Next, we check whether the automaton $A'$ is deadlock-free. Note that checking whether a state of $A'$ is a deadlock state is in PSPACE: PSPACE is closed under negation, and to show that a state is not a deadlock state, it suffices to guess a path from the state to an accepting state, and this can be done in PSPACE by a routine argument. Now, to show that $A'$ is not deadlock-free, we simply guess a state, and show it be reachable as well as to be a deadlock state. Consequently, checking deadlock-freedom of $A'$ is in PSPACE.

Finally, we need to show that $L(A')$ and $L(G)$ are identical. Consider the complement automaton $\bar{A}$ for $L(G)$ (we do not actually build $\bar{A}$, but compute its states using the subset construction as we need them). We then need to know whether $L(A') \cap L(\bar{A})$ is empty or not. If it is, then $A'$ realizes $L(G)$. If not, then $L(G)$ is not safely realizable. Since each state of $\bar{A}$ requires exponential size to encode, we can determine whether $L(A') \cap L(\bar{A})$ is nonempty in EXPSPACE by guessing an accepting path in each automaton. □

**Theorem 4.** *Checking safe realizability of a bounded MSC-graph is* PSPACE*-hard*.

**Proof.** We reduce the PSPACE-complete problem of determining whether a given NFA, $A$, accepts $\Sigma^*$, to checking safe realizability. Assume $A = \langle Q, \Sigma, \delta, q_{\text{init}}, l, F\rangle$ is $\Sigma$-labeled on states rather than on transitions, i.e., $l : Q \mapsto \Sigma$, and $\delta \subseteq Q \times Q$. Let $\Sigma = \{a_1, \ldots, a_k\}$ and $Q = \{q_1, \ldots, q_n\}$. We build from $A$ an MSC graph $G$, which will have nodes $(Q' = \{q'_1, \ldots, q'_n\}) \cup \{start, left, right\} \cup (V = \{v_{a_1}, \ldots, v_{a_k}\})$. The edges between vertices in $Q'$ will be identical to the transition relation $\delta$ over $Q$, and every node $q' \in Q'$ is labeled by an MSC with one synchronous (acknowledged) message from process $P_1$ to $P_2$, where the content of the message is $l(q)$. The vertices $V$ will form a complete subgraph and $v_a \in V$ is labeled by an MSC where $P_1$ sends $a$ (synchronously) to $P_2$. The start node *start* is labeled with the empty MSC. It has edges to both the *left* and *right* nodes. The node *left* is labeled by an MSC where $P_3$ sends the (synchronous) message "left" to $P_2$ and the (synchronous) message "go" to $P_1$. The node *right* is labeled by an MSC where $P_3$ sends the (synch) message "right" to $P_2$ and the (synch) message "go" to $P_1$. The *right* node has an edge to the initial state $q'_{\text{init}} \in Q'$. The *left* node has edges to all vertices in $V$. The terminal nodes of $G$ are all nodes of $V$ as well as those nodes $q' \in Q'$ such that $q \in F$.

We claim that $L(G)$ is safely realizable iff $L(A) = \Sigma^*$. Let us consider whether the product $\Pi A_i$ of the component automata $A_i$, each associated with corresponding process $P_i$, can deadlock. The component automaton $A_1$ waits for the message "go" from $P_3$, acks it, and then synchronously sends all possible message sequences to $P_2$. (Note that this is so because the projection of $V$ onto process $P_1$ accepts all possible strings following a "go".)

Suppose $L(A) = \Sigma^*$. Then, the automaton $A_2$ waits for the message "left" or "right" from $P_3$, and then can receive every possible message from $P_1$ (each message is acknowledged). Hence the product $\Pi A_i$ is a safe realization. If $L(A) \neq \Sigma^*$, then let $w$ be a string such that $w \notin L(A)$. Suppose $P_3$ sends the "right" instruction to $P_2$ and "go" to $P_1$. After $P_1$ receives the "go" instruction from $P_3$, let $P_1$ attempt to send $w$ to $P_2$ (by moving through the *left* node into the $V$ component). Since $w \notin L(A)$, there is some prefix $w'$ of $w$ such that, after $P_1$ sends $w'$ there is no execution of $P_2$ (necessarily in the $Q'$ component) that receives it, i.e., no path for $P_2$ which after receiving the "right" instruction receives $w'$. Hence $L(G)$ is not safely realizable. □

## 4. Verification

Now we turn our attention to the verification problem where the system to be verified is described by an MSC-graph $G$. We will consider two semantics for the verification problem, the set $L(G)$ of all the MSCs specified by $G$, and the set $L^w(G)$ of all the MSCs in the weak closure. First suppose the specification is given by an automaton $A$ accepting linearizations corresponding to "bad" behaviors. If the semantics of an MSC-graph $G$ is $L(G)$, then the verification problem, namely, checking emptiness of $L(G) \cap L(A)$, is undecidable in general and PSPACE-complete for bounded graphs [3]. As our results will indicate, when the semantics of $G$ is $L^w(G)$, the verification problem is undecidable even for bounded graphs. Since MSCs specify partially ordered executions, we proceed to consider partial-order specifications.

### 4.1. MSC membership

Given MSC graph $G$ and given an MSC $M$, we wish to know (1) is $M \in L(G)$? and (2) is $M \in L^w(G)$? There are at least two reasons to consider this problem. First, $M$ may specify an undesirable scenario, so a positive answer to any of these two questions imply existence of a bug. Second, $M$ may specify a desired behavior, and answering these questions can help avoid redundancy.

As discussed earlier, we can equate an MSC over $k$ processes with a "well-formed" $k$-tuple $\langle s_1, \dots, s_k \rangle$ of strings $s_i$, where $s_i$ indicates the linearly ordered sequence of messages sent and received by process $i$.

First we consider the question of checking if a given MSC $M$ belongs to $L(G)$. There are two cases to this question depending on whether the number of processes $k$ in the MSCs is fixed or not. We observe that for a fixed number of processes, $k$, the question can be answered in time $O(n^{2k})$, and we show that for an arbitrary number of processes the question is NP-complete. Boundedness is not relevant to these results.

**Theorem 5.** *Given an MSC-graph G and an MSC M over k processes, there is an algorithm that decides in $O(|G||M|^k)$ time whether $M \in L(G)$.*

**Proof.** Let $M = \langle s_1, \dots, s_k \rangle$. Let $s_i[j, j']$ denote, for $0 \leqslant j \leqslant j' \leqslant |s_i|$, the substring of $s_i$ starting at position $j$ and ending at position $j'$. Since $k$ is fixed, we can build a graph $H$ whose nodes are $(v, \bar{d})$, where $v$ is a node of $G$ and where $\bar{d} = (d_1, \dots, d_k)$, $0 \leqslant d_i \leqslant |s_i|$. There will be an edge $(v, \bar{d}) \rightarrow (v', \bar{d}')$ in $H$ if and only if there is an edge $v \rightarrow v'$ in $G$ and $\mu(v') = \langle s_1[d_1+1, d_1'], \dots, s_k[d_k+1, d_k'] \rangle$. We mark an initial node of $H$, namely the node $(v_{\text{init}}, \bar{d})$ such that $v_{\text{init}}$ is the initial node of the MSC graph, and $\mu(v_{\text{init}}) = \langle s_1[0, d_1], \dots, s_k[0, d_k] \rangle$. (If this initial node does not exist, then we already know $M \notin L(G)$.) Now, $M \in L(G)$ iff $(v, (|s_1|, \dots, |s_k|))$ is reachable for some terminal vertex $v$ of $G$. The size of $H$ is at most $O((|G||M|^k))$, hence we can compute reachability in that time bound. Note that we need not actually construct $H$, but can compute reachability on it on the fly, computing nodes only as needed. □

Next we show NP-completeness for the membership problem. Our proof is very similar to the proof given by [14] for "template matching" in MSC graphs, but because template matching offers more flexibility than finding a given MSC, we need a reduction from a slightly different NP-complete problem. The result can also be derived from an earlier result on membership problems for trace languages [6]. We give our explicit proof, because it also yields the stronger facts that the result remains true for complete MSC-graphs and acyclic MSC-graphs.

**Theorem 6.** *Given an MSC-graph G and an MSC M, it is NP-complete to determine if $M \in L(G)$, even when G is a complete graph, or when G is an acyclic graph.*

**Proof.** The problem is contained in NP because we can guess a path in $G$ and easily verify that the path generates $M$.

To show NP-hardness, we provide a reduction from the NP-complete problem *ONE-IN-THREE-3-SAT* [16]: given a 3-CNF formula $\varphi$, is there a satisfying assignment to the variables such that each clause of $\varphi$ gets *exactly one literal* assigned `true`? From a 3CNF formula $\varphi = C_1 \wedge \dots \wedge C_m$, over variables $x_1, \dots, x_n$, we define an MSC graph $G$ and an MSC $M$ over $2m + 2n$ processes. The underlying graph of $G$ is a complete graph, and $M$ does not depend on $\varphi$. For each clause $C_j$, we have two processes $P_{j,1}$ and $P_{j,2}$, and for each variable $x_i$, we have two processes $Q_{i,1}$ and $Q_{i,2}$. The complete graph $G$ has $2n$ vertices $V = \{v_i, w_i \mid 1 \leqslant i \leqslant n\}$, where $n$ is the number of variables in $\varphi$. All vertices of $G$ are initial vertices. For each $i$ we label $v_i$ by an MSC $M_{x_i}$ in which there is one message (labeled, say, $a$) from process $P_{j,1}$ to $P_{j,2}$ precisely when variable $x_i$ appears positively in $C_j$. In addition, there is a message $a$ sent from $Q_{i,1}$ to $Q_{i,2}$ in $M_{x_i}$. Likewise, $w_i$ is labeled by an MSC $M_{\bar{x}_i}$, which does the opposite of $M_{v_i}$: there is one message labeled $a$ from process $P_{j,1}$ to $P_{j,2}$ when variable $x_i$ appears negatively. Again, in addition, the message $a$ is sent from $Q_{i,1}$ to $Q_{i,2}$.

Finally, we define $M$, which does not depend on $\varphi$. In $M$, for each $j$, there is one $a$-message sent from $P_{j,1}$ to $P_{j,2}$, and for each $i$ there is an $a$-message sent from $Q_{i,1}$ to $Q_{i,2}$. It is not difficult to see that $M \in L(G)$ iff there is a satisfying assignment to $\varphi$ that sets precisely one literal in each clause to true. $\square$

Now we consider the membership question for weak-closure semantics: is $M \in L^w(G)$? This problem turns out to be much easier:

**Theorem 7.** *Given an MSC-graph $G$ and an MSC $M$, there is an algorithm that in time $O(|G||M|)$ determines whether $M \in L^w(G)$.*

**Proof.** Suppose $G$ and $M = \langle s_1, \ldots, s_k \rangle$ are defined over $k$ processes. Checking whether $M \in L^w(G)$ amounts to simply checking whether, for each process $i$, $s_i$ can be generated by the automaton given by the "projection" of $G$ onto process $i$ (see [1]).

For each process $i$, let $G_i$ be the projection of $G$ onto the events of process $i$: $G_i$ is like $G$, but each vertex $v$ in $G_i$ is labeled with the projection onto process $i$ of the MSC labeling $v_j$ in $G$. The accept states of $G_i$ are the same as those of $G$. $G_i$ can be viewed as an ordinary automaton over the alphabet of events belonging to process $i$. Then $M \in L^w(G)$ iff $s_i \in G_i$ for each $i$. Building $G_i$'s can be done in linear time, and checking whether $s_i \in G_i$ can be done in time $O(|G_i||s_i|)$, for each $i \in [k]$. Thus, the total time is $O(|G||M|)$. $\square$

## 4.2. Checking local properties

Given $G$, we want to know whether $L^w(G)$ satisfies a property $\varphi$. A property $\varphi$ is linearization independent if it will hold for one linearization of an MSC iff it holds for all. A property $\varphi$ of MSCs is said to be *local* if it (syntactically) only refers to events on one process, $P_i$. Such local properties are clearly linearization independent, because every linearization of an MSC preserves the local order of events on a given process. Boolean combinations of linearization independent properties are also clearly linearization independent. In order to make this a precise definition, we need to fix a specification logic. However, our upper bound results are applicable irrespective of the particular choice as long as checking whether a particular word satisfies a property can be done in polynomial-time, and our negative results use properties of a very limited form such as "event $x$ eventually occurs on process $P_i$", or "event $x$ never occurs on process $P_i$". As an example of such a specification logic, consider the following: a property is a Boolean combination of atomic properties, and an atomic property is a regular language over the events of a single process, specified by a deterministic finite automaton. Another example is the restricted temporal logic $TLC^-$ interpreted over partially ordered structures [15].

**Theorem 8.** *There are local properties $\varphi_1$ and $\varphi_2$ such that for a finite MSC set $L$, it is coNP-complete to determine if every MSC in $L^w$ satisfies $\varphi_1 \vee \varphi_2$.*

**Proof.** The problem is in coNP, because we can guess projections on each process, check that they combine into a valid MSC, and then check that the respective local strings satisfy the simple eventuality described by the local properties $\neg\varphi_1$ and $\neg\varphi_2$.

The hardness proof is a reduction from 3SAT. Let $\Gamma = \langle C_1, \ldots, C_m \rangle$ be the clauses of the 3SAT formula, ordered in some arbitrary way, and let $x_1, \ldots, x_n$ be its variables. We will add new variables $y$ and $z_1, \ldots, z_m$. Our new ordered list of clauses will be $\Delta = \langle y \vee \neg z_1, y \vee \neg z_2, \ldots, y \vee \neg z_m, C_1 \vee z_1, C_2 \vee z_2, \ldots, C_m \vee z_m \rangle$. Clearly, $\Gamma$ has a satisfying assignment iff $\Delta$ has a satisfying assignment with $y = 0$. Let $\Delta|_1^k$ denote the first $k$ clauses in the list $\Delta$. Notice that for every clause $C_i$ and for every assignment to variables occurring in $C_i$, there is a satisfying assignment of $\Delta|_1^{i+m-1}$ which agrees with that assignment (just set $y = 1$, and $z_j = 1$ for $j < i$. Now you are free to set the variables in $C_i$ in whichever way).

Now we are ready to describe our MSC set, and our properties $\varphi_1$ and $\varphi_2$. The MSC set will consist of one process for every variable and every clause in $\Delta$. In addition, there will be an extra process called $P_f$, which will serve to tabulate whether the formula has been satisfied or not. There will be one MSC, $M_t$ based on the “trivial” satisfying assignment to $\Gamma$, namely $P_y$ will send *true* to every clause that contains it, in their lexicographical order. Likewise, the $P_{z_i}$’s will send *true* to every clause that contains $z_i$, respectively. The $x_i$ variables, can either send *true* or *false* to their clauses, it does not matter here. Each clause process $P_{C'}$, after receiving its truth assignment in messages (which it reads in the lexicographical order of the variables), then receives a message from its predecessor clause (if there is one) which either indicates that the prior clauses have all been satisfied or not. If the prior clauses have been satisfied, and if $C'$ itself has also been satisfied, then $C'$ propagates the “satisfied” message to the next clause in the ordered list. Otherwise, it propagates “not satisfied”. The last clause $C''$ propagates this message to $P_f$, which does nothing other than to receive it. Clearly, for the assignment on which $M_t$ is based, $P_f$ will receive a satisfied message.

Next, for each clause $C_i \vee z_i$, and for each satisfying assignment $\rho$ to the variables in $C_i \vee z_i$ (there are only a constant number of these, since $C_i$ is a 3CNF clause), we will add a new MSC $M_{C_i,\rho}$ which mimics the same thing as above, only the assignment to the variables is one consistent with both $\rho$ and the assignment mentioned above which satisfies $\Delta|_1^{i+m-1}$. Finally, we add another MSC $M_y$, whose only purpose is to exhibit one MSC such that the “assignment” to $y$ is *false*.

Our set $L$ of MSCs contains $M_t$, $M_y$ and the MSCs $M_{C_i,\rho}$. Consider an MSC $M \in L^w$. If $P_f$ receives a “satisfied” message, then we can construct a satisfying assignment for $\Delta$ from $M$. Moreover, if $P_y$ sends *false*’s in $M$, then we can construct a satisfying assignment which assigns $y = 0$, i.e., a satisfying assignment to $\Gamma$. We claim that the converse holds as well, i.e., if there is such a satisfying assignment, then there will be an $M$ weakly-implied by $L$ where $P_y$ sends *false* (call this $\neg\varphi_1$) and $P_f$ receives “satisfied” (call this $\neg\varphi_2$).

| | *finite set* | *bounded graphs* | *unbounded* |
|---|---|---|---|
| weak | coNP-complete | undecidable | undecidable |
| safe | P-time | EXPSPACE-complete | undecidable |

Fig. 5. Summary of results on realizability.

To see this, note that, locally, each variable process sees both assignments to that variable in some MSC in $L$. Moreover, locally each clause process sees every satisfying assignment to that clause. Now, if there is global satisfying assignment that sets $y$ to *false*, then it must be possible to construct it by combining the local satisfying assignment in a consistent way, i.e., by having an implied MSC $M \in L$ which exhibits this satisfying assignment. □

It follows that checking whether every MSC in $L^w(G)$, for an acyclic MSC-graph $G$, satisfies a boolean combination of local properties is coNP-complete.

**Theorem 9.** *There is a boolean combination $\varphi$ of local properties, such that given a bounded MSC-graph $G$, it is undecidable to check if every MSC in $L^w(G)$ satisfies $\varphi$.*

**Proof.** The proof uses precisely the same complete MSC graphs given in the proof of Theorem 1, which reduce an instance of RPCP to checking whether $L(G) = L^w(G)$. Note that in that setting, if there is an implied but unspecified MSC $M$, that is, if there is a solution to the RPCP, then by the construction, there is a solution in which the two strings use exclusively words from different lists. In the notation of the proof of Theorem 1, all $a_i$ bits are 0, and all $b_i$ bits are 1, that is, in the implied MSC $M$, every message sent by the process $P_1$ to $P_2$ is of the form $(i, 1)$ (let's call this property $\psi_1$) and every message sent by the process $P_4$ to $P_3$ is of the form $(i, 0)$ (let's call this property $\psi_2$). Conversely, if there is an implied MSC that satisfies the property $\psi_1 \wedge \psi_2$, then RPCP has a solution. Hence, RPCP has no solution iff every MSC in $L^w(G)$ satisfies the property $\neg\psi_1 \vee \neg\psi_2$. Clearly, $\psi_1$ and $\psi_2$ are local properties ($\psi_1$ depends only on the events of process $P_1$, and $\psi_2$ on the events of process $P_4$). □

## 5. Conclusions

We have studied various algorithmic questions related to checking realizability and verifying MSC-graphs and bounded MSC-graphs. A subsequent recent paper by Lohrey solves the two gaps in our results [11]: checking safe realizability for bounded HMSCs is EXPSPACE-hard, and is undecidable in the general case. The table in Fig. 5 summarizes the computational complexity of various realizability problems. Note that our undecidability proof for weak realizability applies only in the setting with FIFO communication architecture, and a recent result by Morin [12] establishes decidability of weak realizability of bounded MSC-graphs under a non-FIFO architecture.

## Acknowledgements

Thanks to Remi Morin for comments that helped clarify some ambiguity about our assumed communication architecture in Theorem 1. Thanks to an anonymous referee for pointing out the work of [6], and its implications for Proposition 6.

## References

[1] R. Alur, K. Etessami, M. Yannakakis, Inference of message sequence charts, in: Proc. 22nd Internat. Conf. on Software Engineering (ICSE), 2000, pp. 304–313.
[2] R. Alur, G.J. Holzmann, D.A. Peled, An analyzer for message sequence charts, Software Concepts Tools 17 (2) (1996) 70–77.
[3] R. Alur, M. Yannakakis, Model checking of message sequence charts, in: Proc. Tenth Internat. Conf. on Concurrency Theory (CONCUR), Lecture Notes in Computer Science, Vol. 1664, 1999, pp. 114–129.
[4] H. Ben-Abdallah, S. Leue, Syntactic detection of process divergence and non-local choice in message sequence charts, in: Proc. Third Internat. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science, Vol. 1217, 1997, pp. 259–276.
[5] H. Ben-Abdallah, S. Leue, MESA: support for scenario-based design of concurrent systems, in: Proc. Fourth Internat. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science, Vol. 1384, 1998, pp. 118–135.
[6] A. Bertoni, G. Mauri, N. Sabadini, Membership problems for regular and context-free trace languages, Inform. Comput. 82 (2) (1989) 135–150.
[7] G. Booch, I. Jacobson, J. Rumbaugh, Unified Modeling Language User Guide, Addison-Wesley, Reading, 1997.
[8] J. Henriksen, M. Mukund, K. Narayan Kumar, P.S. Thiagarajan, On message sequence graphs and finitely generated regular MSC languages, in: Proc. 27th Internat. Coll. on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, Vol. 1853, 2000, pp. 675–686.
[9] G.J. Holzmann, D.A. Peled, M.H. Redberg, Design tools for requirements engineering, Lucent Bell Labs Tech. J. 2 (1) (1997) 86–95.
[10] ITU-T recommendation Z.120. Message Sequence Charts (MSC'96), 1996.
[11] M. Lohrey, Safe realizability of high-level message charts, in: Proc. 13th Internat. Conf. on Concurrency Theory (CONCUR), 2002.
[12] R. Morin, Recognizable sets of message sequence charts, in: Proc. 19th Annual Symp. on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science, Vol. 2285, 2002, pp. 523–534.
[13] A. Muscholl, D.A. Peled, Message sequence graphs and decision problems on Mazurkiewicz traces, in: Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science (MFCS), 2000, pp. 81–89.
[14] A. Muscholl, D.A. Peled, Z. Su, Deciding properties of message sequence charts, in: Proc. First Internat. Conf. on Foundations of Software Science and Computation Structures, 1998, pp. 226–242.
[15] D.A. Peled, Specification and verification of message sequence charts, in: Proc. IFIP Internat. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII), 2000, pp. 139–154.
[16] T.J. Schaefer, The complexity of satisfiability problems, in: Proc. Tenth ACM Symp. on Theory of Computing (STOC), 1978, pp. 216–226.