Realizability and Verification of MSC Graphs

Rajeev Alur[1,2], Kousha Etessami[2], and Mihalis Yannakakis[2]

[1] Dept. of Computer and Info. Science, U. of Pennsylvania
[2] Bell Labs, Lucent Technologies
Emails: {alur,kousha,mihalis}@research.bell-labs.com

Abstract. Scenario-based specifications such as message sequence charts (MSC) offer an intuitive and visual way of describing 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 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 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 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 solution. For temporal logic specifications, considering the closure makes the verification problem harder: while checking LTL properties of $L(G)$ is PSPACE-complete and checking *local* properties has polynomial-time solutions, even for boolean combinations of local properties of $L^w(G)$, verifying acyclic graphs is coNP-complete and verifying bounded graphs is undecidable.

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 [12], and incorporated in modern software engineering notations such as UML [5]. 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 [9], detecting non-local choice [6], and model checking [3], and tools such as uBET [8] and MESA [4].

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 potentially unbounded number of messages yet to be received [9, 3]. 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, 4]. The boundedness property of an MSC-graph can be checked in time exponential in the number of processes [3]. 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. 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 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. 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 [10]. In this paper, we show that under the closure-semantics, the model checking questions become harder: for an acyclic graph (or even a set of MSCs) the problem is coNP-complete, and for bounded graphs the problem is undecidable, even for simple linearization-invariant local specifications.

2 MSCs and MSC Graphs

We start by recalling the definition of message sequence charts. Our definition captures the essence of the ITU standard MSC'96, and is analogous to the definitions of labeled MSCs given in [2, 3]. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of processes, and Σ be a message alphabet. We write $[n]$ for $\{1, \ldots, n\}$. 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 $\Sigma^S = \{send(i, j, a) \mid i, j \in [n] \;\&\; a \in \Sigma\}$ of *send labels*, the set $\Sigma^R = \{receive(i, j, a) \mid i, j \in [n] \;\&\; a \in \Sigma\}$ of *receive labels*, and $\Sigma = \Sigma^S \cup \Sigma^R$ as the set of *event labels*. A *Σ-labeled MSC M* over processes $\mathcal{P}$ is given by:

1. a set E of events which is partitioned into a set S of "send" events and a set R of "receive" events;

2. a mapping $p : E \mapsto [n]$ that maps each event to a process on which it occurs;
3. a bijective mapping $f : S \mapsto R$ between send and receive events, matching each send with its corresponding receive;
4. a mapping $l : E \mapsto \Sigma$ 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 $p(f(s)) = j$;
5. for each $i \in [n]$, a total order $\leq_i$ on the events of process P_i, that is, on the elements of $p^{-1}(i)$, such that the transitive closure of the relation

$$\leq \doteq \cup_{i\in[n]} \leq_i \cup \{(s, f(s)) \mid s \in S\}$$

is a partial order on E.

We require our MSCs to satisfy an additional *non-degeneracy* condition: there is no reversal of the order in which two *identical* messages sent by some process P_i are received by another process P_j. Observe that the information in such MSCs can be captured by a word over Σ that corresponds to the sequence of labels of a linearization consistent with the partial order $\leq$. To be precise, a word w over Σ is *well-formed* if all receive events have matching sends; and is *complete* if all send events have matching receives. Then, a word w over the alphabet Σ is a linearization of an MSC iff it is well-formed and complete (see [1]).

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^I, a terminal vertex v^T, and a labeling function μ that maps each vertex v to an MSC. The paths that start at the initial vertex and end at the terminal vertex represent 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 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 linearizations of the MSCs in it.

In general, the set $L(G)$ is not regular. The problematic case is 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 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 ρ of G, the subgraph

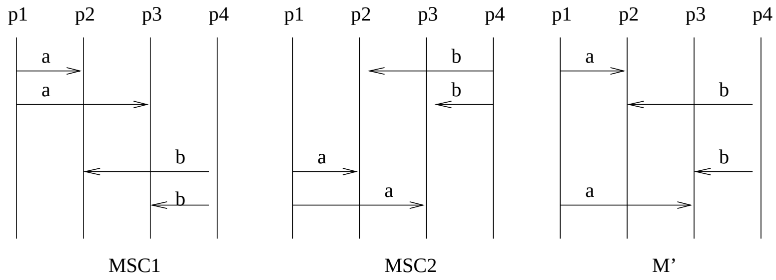

Fig. 1. Weak Inference

of the communication graph H_ρ induced by the set P_ρ of active processes of the cycle is strongly connected. In other words, communication graph H_ρ 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 [7].

3 Realizability

3.1 Weak Realizability

Consider the two MSCs MSC1 and MSC2 shown in Figure 1. 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, 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 $i \in [n]$, there exists an MSC $M_i \in L$ such that $M|_i = M_i|_i$. 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 notion of weak realizability was introduced in [1] in the context of the underlying communicating state machines that would realize the communication patterns specified by given MSCs. In particular, we had shown that L is weakly realizable iff there exist automata A_i, one per process P_i, asynchronously communicating with one another over FIFO buffers, such that the global product of these automata exhibits precisely the behaviors in L.

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. We show that checking weak realizability is undecidable for bounded graphs.

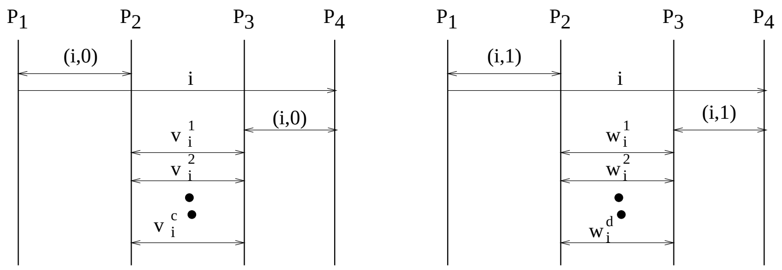

Fig. 2. MSCs M_i^0 and M_i^1

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 $\langle(v_1, w_1), (v_2, w_2), \dots, (v_r, w_r)\rangle$, where $v_i, w_i \in \Sigma^*$, for some fixed finite alphabet Σ, with designated initial pair (v_1, w_1), determine whether there is a sequence of indices $i_2, \dots, i_m$, such that

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

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 $i_2 \dots i_m$, such that equality 1 holds, and furthermore, for all $j \leq m$, the string $w_1 w_{i_2} \dots w_{i_j}$ is a prefix of the string $v_1 v_{i_2} \dots v_{i_j}$ (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 $\{(v_1, w_1), (v_2, w_2), \dots, (v_r, w_r)\}$, determine whether there are indices $i_1, \dots, i_m$ such that $x_{i_1} \dots x_{i_m} = y_{i_1} \dots y_{i_m}$, 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 $j \leq m$, $y_{i_1} \dots y_{i_j}$ is a prefix of $x_{i_1} \dots x_{i_j}$. The proof that RPCP is undecidable is omitted due to lack of space.

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 MSC's, 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 MSC's 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 $\Delta = \{(v_1, w_1), \dots, (v_m, w_m)\}$ 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 Figure 2. Thus in M_i^0, process 1 sends synchronously

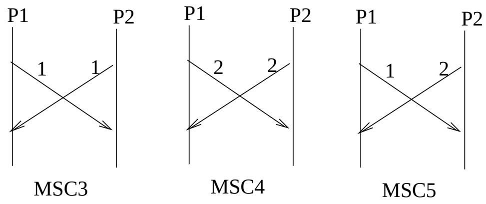

Fig. 3. Safe realizability

$(i, 0)$ 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. $\Delta \in RPCP$ iff L^* is not weakly realizable.

□

3.2 Safe Realizability

As a motivation for safe realizability, consider the MSCs in Figure 3. 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.

Safe realizability detects such problems by considering *partial MSCs*, or equivalently, prefixes of linearizations. Let $pref(L)$ denote the set of prefixes of the MSCs or words in L. Then, a set L of MSCs *safely implies* a well-formed word w (i.e. a partial MSC) iff for all $i \in [n]$, there exists a word $v_i \in pref(L)$ such that $w|_i = v_i|_i$. The set L is safely realizable iff $pref(L)$ contains all the words safely implied by L. As shown in [1], L is safely realizable iff there exist automata A_i, one per process P_i, asynchronously communicating with one another over FIFO buffers, such that the global product of these automata is deadlock-free and exhibits precisely the behaviors in L.

The MSC-graph G is said to be safely realizable if the set $L(G)$ of MSCs is. For a finite set of MSCs, it is known that weak realizability is coNP-complete while safe realizability has polynomial-time solution. 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, but its precise complexity class remains an open problem.

Theorem 2. *Checking safe realizability of a bounded MSC-graph is in* EXPSPACE.

Proof. We know that $L(G)$ is definable by an exponential sized automaton A each of whose states can be encoded in polynomial space. Likewise, we can build a concurrent product $A' = \Pi_i A_i$, where the A_i are the local automata formed by the projection of G onto process i, and then determinized and minimized. If $L(G)$ is safely realizable, then we know that A' is such a realization. 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. We can then "build" the complement automaton $\bar{A}$ for $L(G)$ (we don't 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. $\square$

Theorem 3. *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 Σ^*, to checking safe realizability. Assume $A = \langle Q, \Sigma, \delta, q_0, l, F\rangle$ is Σ-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 $(Q' = \{q'_1, \ldots, q'_n\}) \cup \{\mathit{start}, \mathit{left}, \mathit{right}\} \cup (V = \{v_{a_1}, \ldots, v_{a_k}\})$. The edges between vertices in Q' will be identical to the transition relation δ 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 $q'_0 \in Q'$. The *left* node has edges to all vertices in V. The target nodes of G are all nodes of V as well as those nodes $q' \in Q'$ such that $q \in F$. The claim is that $L(G)$ is safely realizable iff $L(A) = \Sigma^*$. □

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 [1]. 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 4. *Given an MSC-graph G and an MSC M over k processes, there is an algorithm that decides in $O(n^{2k})$ time whether $M \in L(G)$, where n is the size of the input (G, M).*

Proof. Let $M = \langle s_1, \dots, s_k \rangle$. Let $s_i[j, j']$ denote, for $0 \le j \le j' \le |s_i|$ the substring of s_i starting at position j and ending at position j'. Since k is fixed, we can build, by dynamic programming, a $2k$-dimensional array of boolean values

K, where $K[d_1, \ldots, d_k, d'_1, \ldots, d'_k]$, is true for $0 \leq d_i \leq d'_i \leq |s_i|$, whenever $\langle s_1[d_1, d'_1], \ldots, s_k[d_k, d'_k]\rangle \in L(G)$. K can easily be computed in time $O(n^{2k})$. Thus, $M \in L(G)$ iff $K[0, \ldots, 0, |s_1|, \ldots, |s_k|]$ is true. $\square$

Next we show NP-completeness for the membership problem. The proof is very similar to the proof given by [9] 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.

Theorem 5. *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* [11]: given a 3-CNF formula φ, is there a satisfying assignment to the variables such that each clause of φ gets *exactly one literal* assigned `true`. From a 3CNF formula $\varphi = C_1 \wedge \ldots \wedge C_m$, over variables $x_1, \ldots, x_n$, 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 φ. 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 $V = \{v_i, w_i \mid i \in [n]\}$, where n is the number of variables in φ. All vertices of G are initial vertices. For each i we label v_i by an MSC M_{x_i} in which there is a 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 φ. 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 φ 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 6. *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. 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. 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 φ. A property φ is linearization independent if it holds for one linearization of an MSC iff it holds for all. A property φ of MSCs is said to be *local* if it (syntactically) only describes events on one process. Boolean combinations of local properties are clearly linearization independent. It is known that checking linearization independent properties of $L(G)$ can be done in polynomial-time [10]. However, the following theorems shows that we cannot hope to find efficient model checking algorithms for $L^w(G)$.

Theorem 7. *There are local properties φ_1 and φ_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 membership in co-NP is obvious. 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, Γ has a satisfying assignment iff Δ has a satisfying assignment with $y = 0$. Let $\Delta|_1^k$ denote the first k clauses in the list Δ. 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.

Now we are ready to describe our MSC set, and our properties φ_1 and φ_2. The MSC set will consist of one process for every variable and every clause in Δ. 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 Γ, 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 doesn't 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 ρ 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 ρ and the assignment mentioned above which satisfies $\Gamma|_1^{i+m-1}$. 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 Δ 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 assingment to Γ. 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 $\neg\varphi_2$). $\square$

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

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 there is an implied but unspecified MSC (and thus a solution to the RPCP) if and only if every message sent by process P_1 to P_2 is of the form $(i, 0)$, while every message sent by P_3 to P_4 is of the form $(i, 1)$, or vice versa. $\square$

Acknowledgements. This research was partially supported by NSF Career award CCR97-34115, NSF award CCR99-70925, and Sloan Faculty Fellowship.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proc. of 22nd Int. Conf. on Software Engineering*, 2000.

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

[3] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Concurrency Theory, Tenth Intl. Conference*, LNCS 1664, pages 114–129, 1999.

[4] H. Ben-Abdallah and S. Leue. MESA: Support for scenario-based design of concurrent systems. In *Proc. 4th TACAS*, LNCS 1384, pages 118–135, 1998.

[5] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide*. Addison Wesley, 1997.

[6] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Proc. 2nd TACAS*, 1997.

[7] J. Henriksen, M. Mukund, K. Narayan Kumar, and P.S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proc. 27th ICALP*, LNCS 1853, pp. 675–686, 2000.

[8] G.J. Holzmann, D.A. Peled, and M.H. Redberg. Design tools for for requirements engineering. *Lucent Bell Labs Technical Journal*, 2(1):86–95, 1997.

[9] A. Muscholl, D. Peled, and Z. Su. Deciding properties of message sequence charts. In *Foundations of Software Science and Computation Structures*, 1998.

[10] D. Peled. Specification and verification of message sequence charts. In *Proc. IFIP FORTE/PSTV*, 2000.

[11] T.J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th ACM Symp. on Theory of Computing*, pages 216–226, 1978.

[12] ITU-T recommendation Z.120. Message Sequence Charts (MSC'96), 1996.