# SPECIFICATION AND VERIFICATION OF MESSAGE SEQUENCE CHARTS

Doron Peled
*Bell Labs*
*Murray Hill, NJ 07974*
doron@research.bell-labs.com

**Abstract** The use of message sequence charts (MSCs) is popular in designing and documenting communication protocols. A recent surge of interest in MSCs has led to various algorithms for their automatic analysis, e.g., finding race conditions. In this paper we adopt a causality based temporal logic to specify properties of MSCs. This alleviates some problems that arise when specifying properties of MSCs using the traditional interleaving-based linear temporal logic: systems of MSCs are not necessarily finite state systems, leading to undecidability of LTL model checking. Even when dealing with finite state MSC systems, the set of linearizations can easily generate an exponential state space explosion. We provide an efficient model checking algorithm for systems of MSCs. Our construction models the FIFO MSC systems using a restricted version of $\omega$-automata with *two* successor relations. We implemented a model checking environment for MSCs as an extension to the SPIN model checking system.

**Keywords:** Message sequence charts, Model checking, Specification, Verification.

## 1. INTRODUCTION

Software verification is a very challenging task. One of the reasons is that software lacks some of the regularity that is typically found in hardware circuits. Some of the successful attempts to apply verification technology to software focused on communication protocols, abstract versions of algorithms, and finite state systems. One potential target of verification algorithms is the early design of software systems. There, the cost-performance of finding bugs is better than in later development stages, and the description is typically already an abstract version of the desired system.

---

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35533-7_26

Tommaso Bolognesi and Diego Latella (Eds.), *Formal Methods for Distributed System Development.*
© 2000 IFIP International Federation for Information Processing.
Published by Kluwer Academic Publishers. All rights reserved.

*Message Sequence Charts* (MSCs) is a standard ITU notation (see [7]), describing the executions of communication systems. It focuses on the description of messages passed between the participating concurrent processes, and abstracts away most other details, e.g., program variables and their values. Like other ISO (International Standardization Organization) notations, MSCs can be specified both using a visual and a textual form. This simple standard is in particular appealing for specifying communication protocols. A similar notation of 'use cases' is used in object-oriented design. Different collections of *executions* or *scenarios* are described using MSCs, providing a clear representation of different features of systems. Grouping together different scenarios is facilitated using high-level MSCs (HMSCs). These are graphs that include in each of the nodes a single MSC. The graph structure allows describing different alternative executions, according to the different branches of the graph.

In this paper we suggest a generic model checking paradigm that allows the automatic verification of systems of message sequence charts. This paradigm is based on choosing a temporal specification that is based on *partial order semantics*, instead of the traditional *interleaving semantics*. Such a logic cannot distinguish between different linearizations of the same partial order execution, and hence is most suited for causality based models such as MSCs. We provide a concrete suggestion of such a logic, and present a simple implementation of the model checking algorithm, by a translation of the MSC notation and the temporal property to the model checker SPIN [8].

Because HMSCs do not impose any constraint on the number of messages that were sent and were not yet received, HMSCs are not necessarily limited to *finite state systems*. This makes the problem of automatic verification of MSCs tricky. Standard model checking algorithms, which are usually applied to finite state systems [5] fail. Specifically, it was shown that checking linear temporal logic (LTL) properties of HMSCs is undecidable [2](e.g., by a reduction from PCP – post correspondence problem).

Checking LTL properties of MSCs is based on the linearizations of HMSCs, i.e., completing the partial order between events that is imposed by an HMSC scenario into a total order. This approach asserts that *all* the linearizations of an MSC execution satisfy the given LTL specification. To alleviate this problem, one may impose a bound on the number of messages in any communication channel. This allows translating the HMSCs to finite state models that are sometimes exponentially bigger than the original HMSC description. Then it is possible to apply standard model checking algorithms. This approach has the unintuitive

property that different linearizations of the same MSC execution may not agree on satisfying the given LTL specification. Another constraint that makes HMSCs decidable is described in [2, 9]. Besides restricting the systems that can be checked, this approach also requires deciding whether a given system satisfies this constraint, a task which is in the complexity class co-NP complete.

A different approach that is based directly on the partial order semantics of HMSCs is to specify properties of HMSCs using HMSCs. Intersecting two HMSCs is analogous to checking the emptiness of intersecting a system automaton and a property automaton in the automata approach to model checking [11]. The property automaton specifies the disallowed executions. Unfortunately, the problem of emptiness of HMSC intersection turns to be undecidable as well [10] (again by a reduction from PCP).

In the current work, we show how a temporal logic $\mathrm{TLC}^-$, a subset of TLC [1], can be used to assert properties of HMSCs. We provide a model checking algorithm for this logic, which is linear in the size of the checked HMSC, and exponential in the size of the formula and the number of processes. The main observation is that the logic TLC, and its subset $\mathrm{TLC}^-$ are interpreted over the partial order structure of the HMSCs. Hence they do not distinguish between the different linearizations of each partial order execution. Since these formulas can also be interpreted over linearizations of the partial order, a single linearization suffices for each execution. The subset $\mathrm{TLC}^-$ of TLC was carefully chosen. It is capable of asserting interesting properties of communication systems. Moreover, it allows a simple translation from $\mathrm{TLC}^-$ into automata. Yet, one can extend the framework suggested here to other partial order semantics based temporal logics.

## 2. PRELIMINARIES

Each MSC describes a scenario where some processes communicate with each other. Such a scenario includes a description of the messages sent, messages received, the local events, and the ordering between them. In the visual description of MSCs, each process is represented by a vertical line, while a message is represented by a horizontal or slanted arrow from the sending process to the receiving one, as in Figure 1.

**Definition 1** *An MSC $M$ is a sextuple $\langle V, <, L, T, m, \mathcal{P}\rangle$ where $V$ is a set of events, $< \subseteq V \times V$, $\mathcal{P}$ is a set of processes, $L : V \to \mathcal{P}$ is a mapping that associates each event with a process, $T : V \to \{s, r, l\}$ is a mapping that describes the type of each event (send, receive or local, respectively), and $m \subset V \times V$ is a relation between send and receive events. If $(e, f) \in$*

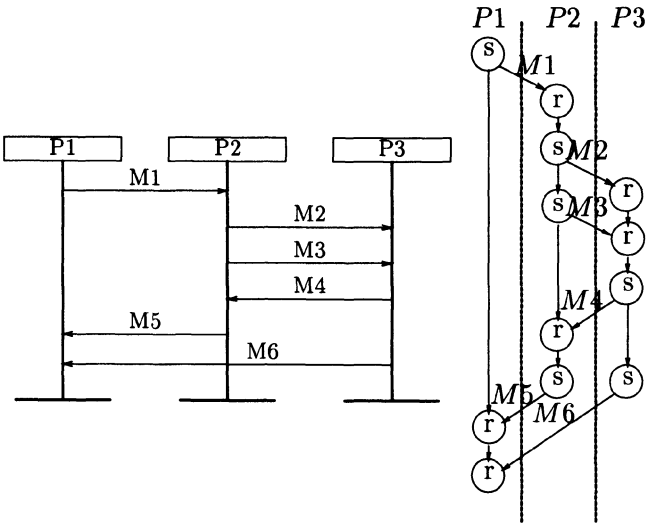


*Figure 1* An MSC and its partial order semantics

*m then* $T(e) = s$, *and* $T(f) = r$. *In this case, we call e and f a* matching pair. *Each send and receive event is paired up using m with exactly one other event. For two events e and f, we have* $e < f$ *when one of the following holds: (1) e and f are a matching pair, or (2) e and f belong to the same process P, with e appearing before f on the process line.*

*We assume* FIFO *(first in first out) message passing, i.e.,* $(T(e_1) = T(e_2) = s \wedge T(f_1) = T(f_2) = r \wedge m(e_1, f_1) \wedge m(e_2, f_2) \wedge L(e_1) = L(e_2) \wedge L(f_1) = L(f_2) \wedge e_1 < e_2) \rightarrow f_1 < f_2$.

Denote by $e \longrightarrow f$ when $e < f$ and either $e$ and $f$ are the *send* and *receive* of the same message, or when they belong to the same process and there is no event between $e$ and $f$ on some process line. That is, $e$ *immediately precedes* $f$. The relation '$<$' is called the *visual ordering* of events and it is obtained from the syntactical representation of the chart (e.g. represented according to the standard syntax ITU-Z120). The transitive closure '$<^*$' of the relation '$<$' is a partial order between the events $V$.

The partial order between the send and receive events of the MSC in Figure 1 is shown in the right side of Figure 1. There, only the 'immediately precedes' order '$\longrightarrow$' is shown. This MSC describes an interaction between three processes, $P1$, $P2$ and $P3$. Process $P1$ sends the message $M1$ to $P2$. After receiving that message, process $P2$ sends two messages, $M2$ and $M3$ to $P3$. After receiving $M3$, process $P3$ sends the message $M4$ back to $P2$ and later also sends the message $M6$ to $P1$.

Process $P2$, after receiving $M4$, sends $M5$ to $P1$. The message $M5$ is received by process $P1$ before the message $M6$. The *send* events of the two messages, $M5$ and $M6$, are unordered.

Some papers (e.g., [3]) distinguish between the above visual order and a more relaxed order among the events that allows for example some receive events of the same process to be unordered. This is because some receive events of messages that are sent from different processes cannot be guaranteed to arrive in a particular order. This relaxed order is used in the MSC tool [3], in conjunction with the visual order, to detect race conditions, where messages may arrive in an order that is not prescribed by the MSC. For example, in the MSC of Figure 1, the message $M6$ may arrive at $P1$ before $M5$, as both are independently sent from different processes. In this paper, we will commit to the simple (visual) order defined above. Next, we define how to sequentially compose a pair of MSCs.

**Definition 2** *The* concatenation *of MSCs* $M_1 = \langle V_1, <_1, L_1, T_1, m_1, \mathcal{P}\rangle$ *and* $M_2 = \langle V_2, <_2, L_2, T_2, m_2, \mathcal{P}\rangle$ *over the same set of processes* $\mathcal{P}$ *and disjoint sets of events* $V_1 \cap V_2 = \emptyset$, *denoted* $M_1\, M_2$, *is* $\langle V_1 \cup V_2, <, L_1 \cup L_2, T_1 \cup T_2, m_1 \cup m_2, \mathcal{P}\rangle$, *where* $<=<_1 \cup <_2 \cup\{(p,q)|L(p) = L(q) \wedge p \in V_1 \wedge q \in V_2\}$

That is, the events of $M_1$ precede the events of $M_2$ for each process, respectively. Notice that there is no synchronization of the different processes when moving from one node to the other. Hence, it is possible that one process is still involved in some actions of a previous node, while another process has advanced to a subsequent node.

Since a communication system usually includes many different scenarios, a high level description is needed for combining them together. This description is represented as a graph, where each node contains one MSC as in Figure 2. Each maximal path in this graph (i.e., a path that is either infinite or ends with a node without outgoing edges), starting from a designated initial state, corresponds to a single *execution* or *scenario.*

**Definition 3** *An HMSC* $N$ *is a triple* $\langle \mathcal{M}, \tau, M_0, \rangle$ *where* $\mathcal{M}$ *is a set of MSCs, all with the same set of processes, and with sets of events disjoint from one another.* $\tau \subseteq \mathcal{M} \times \mathcal{M}$ *is the* edge relation *and the* initial *MSC is* $M_0 \in \mathcal{M}$. *An* execution *of* $N$ *is a path* $\xi = M_0\, M_1\, M_2\, M_3 \ldots$ *of* $N$ *that starts with the initial MSC* $M_0$ *and either ends with a state without outgoing edges, or is infinite.*

Figure 2 shows an example of an MSC graph, where the state in the upper left corner is the starting state. Note that the executions of this

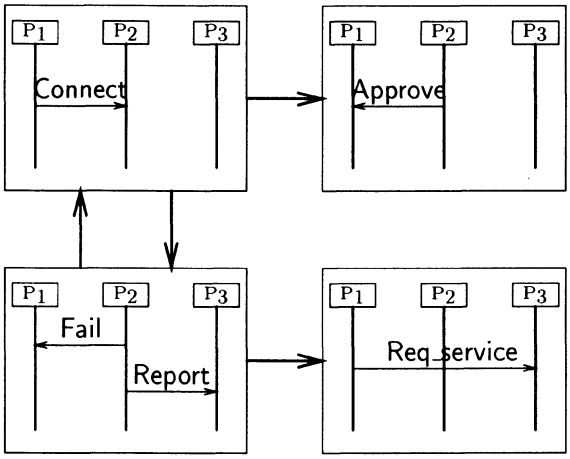


*Figure 2* An HMSC graph

system are either finite or infinite. In Figure 2, process $P2$ may send its Report message after process $P1$ has progressed into the next node (on the left) and has sent its Req_service message.

## 2.1. THE LOGIC TLC$^-$

The logic TLC$^-$ is a subset of the logic TLC [1]. A model of the logic is a finite or infinite partial order $\zeta = (V, <, \longrightarrow)$, where $<\subset V \times V$ is a partial order relation, and $\longrightarrow \subset <$ is the 'immediately precedes' relation. The set of formulas $\mathcal{L}$ of TLC$^-$ over a set of atomic formulas $AP$ is as follows: *true*, *false* $\in \mathcal{L}$, if $p \in AP$, then $p \in \mathcal{L}$, and if $\varphi$, $\psi$ are in $\mathcal{L}$ then $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$, $\exists \bigcirc \varphi$, $\forall \bigcirc \varphi$, $\varphi U \psi$, $\varphi R \psi \in \mathcal{L}$.

An *interpretation function* $I : V \mapsto 2^{AP}$ assigns to each event of $V$ a set of propositions from $AP$. Each proposition in $AP$ represents some property (e.g., of an event, or the local state before or after the event, when the events are taken from some system execution). Then, $I(v)$ returns the set of atomic propositions that hold for $v$. The semantics of the logic is defined as follows.

$(\zeta, v) \models$ *true*.

$(\zeta, v) \models p$ if $p \in I(v)$

$(\zeta, v) \models \varphi \wedge \psi$ if $(\zeta, v) \models \varphi$ and $(\zeta, v) \models \psi$.

$(\zeta, v) \models \neg\varphi$ if it is not the case that $(\zeta, v) \models \varphi$.

$(\zeta, v) \models \exists \bigcirc \varphi$ if for some $w$ such that $v \longrightarrow w$, it holds that $(\zeta, w) \models \varphi$.

$(\zeta, v) \models \varphi U \psi$ if there is a path $v = v_0 \longrightarrow v_1 \longrightarrow \ldots \longrightarrow v_n$, such that $(\zeta, v_n) \models \psi$, and for $0 \leq i < n$, $(\zeta, v_i) \models \varphi$.

We define $false \equiv \neg true$, $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$, $\varphi R \psi \equiv \neg(\neg\varphi U \neg\psi)$, $\forall \bigcirc \varphi \equiv \neg\exists \bigcirc \neg\varphi$. Two additional modalities, $F$ and $G$, can be defined in terms of the previous ones: $F\,\varphi \equiv true U \varphi$, and $G\,\varphi \equiv false R \varphi$. For TLC$^-$ we have selected an *existential until* '$U$' operator, hence its dual *release* '$R$' operator is universal. The full logic TLC contains also a universal *until*, an existential *release*, and a *concurrent with* operator '||'. The modalities $U$ and $R$ satisfy the following equations: $\varphi U \psi \equiv \psi \vee (\varphi \wedge \exists \bigcirc \varphi U \psi)$, $\varphi R \psi \equiv \psi \wedge (\varphi \vee \forall \bigcirc \varphi R \psi)$. An HMSC satisfies a TLC$^-$ specification exactly when each execution of it satisfies the specification.

Some examples for TLC$^-$ Specification are as follows:

$G\,(req \rightarrow F\,ack)$ Every request is causally followed by an acknowledgement.

$G\,(recA \rightarrow \exists \bigcirc sendB)$ A message $B$ is sent immediately after receiving a message $A$.

$\neg\, F\,(tranA \wedge F\,(tranB \wedge F\,tranA))$ Transaction $B$ cannot interfere with the events of transaction $A$.

$G\,(beginA \rightarrow \exists \bigcirc (tranA\,U\,finishA))$ The execution of transaction $A$ is not interrupted by any other event.

## 3. LINEARIZATIONS OF MSCS

In order to simplify model checking, and to reuse code that was written for verifying concurrent protocols, we will interpret TLC$^-$ formulas over linearizations of MSCs. Each linearization is the completion of an MSC execution into a total order. All the linearizations of a single MSC execution must agree with their origin MSC upon the interpretation of each TLC$^-$ formula.

Consider the following linearization of the execution in Figure 1: $(s, P_1)$ $(r, P_2)$ $(s, P_2)$ $(r, P_3)$ $(s, P_2)$ $(r, P_3)$ $(s, P_3)$ $(r, P_2)$ $(s, P_2)$ $(r, P_1)$ $(s, P_3)$ $(r, P_1)$. Each event is denoted as a pair, including the type of event (*send* or *receive*), and the corresponding process.

We also keep with each linearization *forward edges* from each *send* node to its matching *receive*. In the above linearization example, the forward edge from each *send* event always points to the immediate successor on the sequence. However, this is not necessarily always the case. For example (this example was found by Anca Muscholl), consider the case where $P1$ sends a message to $P2$. Before this message is received by $P2$, $P1$ sends a message to $P3$, and then $P3$ sends another message to $P2$,

which arrives before the message that was sent by $P1$. Such cases, which are rare in practical use of MSCs, complicate the verification algorithm that is presented in the sequel.

When the current event is a *send*, there are one or two immediate causal successors according to the order '$\longrightarrow$': an essential successor is the matching *receive* and, unless the current event is the last event for the current process, there is another event on the same process $Q$. Any other type of event (*receive* or *local*) has at most the second kind of immediate successor. The immediate successor of an event in a linearization is not necessarily also a successor under the original partial order relation. For example, the fourth event in the linearization, which is $(r, P_3)$, is immediately followed by $(s, P_2)$. But the latter event is not the immediate successor of the former in the partial order.

The interpretation of a TLC$^-$ formula $\varphi$ over a *linearization* $\xi$ of a partial order $\zeta = (V, <, \longrightarrow)$ follows the interpretation of $\varphi$ in $\zeta$. That is, $(\xi, i) \models \varphi$ if for the $i$th event $v$ of $\xi$, we have that $(\zeta, v) \models \varphi$, according to the above partial order semantics. We add the special event-type propositions *send*, *receive* and *local* with the obvious interpretation, and in addition, a special proposition for each process: we denote the proposition that corresponds to a process $P \in \mathcal{P}$ also by $P$. We also add the special atomic proposition start, which marks nodes that start the linearization of each new HMSC node.

In order to facilitate the translation of TLC$^-$ formulas into automata over sequences, we extend the syntax of TLC$^-$ with three additional types of formulas:

- $\langle P \rangle \varphi$, where $P$ is a process, and $\varphi$ is a TLC$^-$ formula. $(\xi, i) \models \langle P \rangle \varphi$ holds when $(\xi, j) \models \varphi$ for some $j \geq i$, and for each $k$ such that $j > k \geq i$, the $k$th event does not belong to process $P$.

- $\{match\}\varphi$, where $\varphi$ is a TLC$^-$ formula. $(\xi, i) \models \{match\}\varphi$ holds when $(\xi, j) \models \varphi$ for some $j > i$, where the $j$th event of $\xi$ is the matching *receive* for the $i$th event, which is a *send*.

- NEVER $P$, where $P$ is a process. $(\xi, i) \models$ NEVER $P$ holds exactly when for each $j \geq i$, the $j$th event of $\xi$ does not belong to $P$.

For convenience, we assume that for each property $\varphi$ there is some initiating process $Q$. That is, a processes whose first event is the one from which we want to interpret $\varphi$. The first event of process $Q$ cannot be a *receive*. Over linearizations, we can then write $\langle Q \rangle \varphi$. Of course, we can may vary $Q$ over all processes.

# 4. AUTOMATA THEORETIC MODEL CHECKING

A *Büchi automaton* is a sextuple $(S, S_0, \Sigma, L, \delta, F)$, where $S$ is a finite set of states, $S_0 \subseteq S$ are the initial states, $\Sigma$ is the finite alphabet, $L : S \to \Sigma$ is a mapping from the states to the alphabet, $\delta \subseteq S \times S$ is the transition relation (for simplicity of presentation, we deviate from the traditional edge-based labeling and use a state based labeling), and $F \subseteq S$ are the accepting states. A run $\xi$ over a word $a_1 a_2 \ldots \in \Sigma^\omega$ is an infinite sequence of states $s_0 s_1 s_2 \ldots$, with $s_0 \in S_0$, such that for each $i > 0$, $(s_i, \alpha_i, s_{i+1}) \in \delta$. A run is *accepting* if at least one state of $F$ occurs in $\xi$ infinitely many times. A word is accepted by a Büchi automaton exactly when there exists a run accepting it. The *language* $\mathcal{L}a(A)$ of a Büchi automaton $A$ is the set of words that it accepts.

A *specification* automaton $A = (S^A, S_0^A, \Sigma, L^A, \delta^A, F^A)$ is a Büchi automaton that represents a set of *disallowed* executions. Thus, if $\eta$ is some temporal specification of an HMSC system, then the corresponding specification automaton accepts exactly the sequences that do not satisfy $\eta$, i.e., those that satisfy $\neg\eta$. An *implementation* automaton $B = (S^B, S_0^B, \Sigma, L^B, \delta^B, S^B)$ is a restricted version of a Büchi automaton representing the executions of the checked HMSC; it requires all the states $S^B$ to be accepting. Thus, the accepting runs of $B$ are the infinite paths that start from an initial state in $S_0^B$. An implementation automaton represents the modeled system (finite paths are completed to infinite paths for convenience).

The *intersection* (or *product*) $B \times A$ is $(S', S_0', \Sigma, \delta', F')$, where $S' = \{(s,t) | s \in S^A \wedge t \in S^B \wedge L^A(s) = L^B(t)\}$, $S_0' = S' \cap (S_0^A \times S_0^B)$, $\delta' = \{((s,s'),(t,t')) | (s,s') \in S' \wedge (t,t') \in S' \wedge (s,t) \in \delta^A \wedge (s',t') \in \delta^B\}$, $F' = S' \cap (F^A \times S^B)$. We have that $\mathcal{L}a(A \times B) = \mathcal{L}a(A) \cap \mathcal{L}a(B)$.

The intersection of a specification automaton and an implementation automaton gives all the executions of the implementation that are not allowed by the specification. Thus, the intersection is empty exactly if the implementation does not satisfy the specification. Any sequence in the intersection can be used as a counterexample. Checking the emptiness of a Büchi automaton, and in particular the automaton $F'$ that results from the above intersection can be done by applying the DFS algorithm to find reachable maximal strongly connected components of the automaton graph. If there exists such a component that includes at least one state from each accepting subset $f \in F'$, then the automaton is not empty. In particular, in this case there is an *ultimately periodic* sequence of the form $v\,u^\omega$ accepted by $F'$.

## 5. MODEL CHECKING FOR HMSCS

We suggest here a generic framework for model checking HMSCs, with TLC$^-$ as a specific instance of a specification formalism that can be used for this purpose. We first describe a translation from any TLC$^-$ formula $\eta$ into a corresponding Büchi automaton $A$, which accepts the linearizations that satisfy $\eta$. This is a Büchi automaton over linearizations of MSC executions, represented by sequences of events that contain forward edges. The forward edges will introduce another constraint on acceptance, as will be shown later. The construction presented here is in particular appropriate for systems with FIFO asynchronous message passing. There is no fixed dependence relation between the different events according to the process to which they belong, as was the case in previous work, e.g., [1].

We first represent formulas and subformulas in *negation normal form*; in this form, the negation operator can be applied only to atomic propositions. Each TLC$^-$ formula can be converted into an equivalent formula in normal form by repeated use of the DeMorgan laws and the dualities between '$U$' and '$R$', presented above. For example, we can make the following transformations: $\neg\exists \bigcirc (pR(q \vee r)) \equiv \forall \bigcirc \neg(pR(q \vee r)) \equiv \forall \bigcirc (\neg pU\neg(q \vee r)) \equiv \forall \bigcirc (\neg pU(\neg q \wedge \neg r))$.

Due to the existence of forward edges in the linearizations, we will deal with automata with *two* transition relations $(S, S_0, \Sigma, L, \delta, C, F)$, where the additional component $C$ is a relation between pairs of nodes. Thus, the intersection of a specification automaton $(S^A, S_0^A, \Sigma, L^A, \delta^A, C^A, F^A)$ with an implementation automaton $(S^B, S_0^B, \Sigma, L^B, \delta^B, C^B, F^B)$ has an additional component $C'$, which is $C^A \cap C^B \cap (S' \times S')$.

In the construction of the specification automaton, each node in $S$ will contain a *safety component* and a *liveness component.* The safety component is responsible for guaranteeing the consistency between each node and its successors. It contains the following:

- Exactly one of the event-type propositions: *send*, *receive* or *local*, for the type of the current event. In addition, it may include the proposition start.
- A maximal non-contradicting subset of the sub-formulas of the checked property and their negations (in negation normal form). That is, for each subformula $\varphi$, each node will contain either $\varphi$ or $\neg\varphi$, but not both.
- If the node does not contain *never* $P$, then it contains a maximal noncontradicting subset of the formulas of the form $\langle P\rangle\varphi$, where $P$ is a process and $\varphi$ is a subformula of the checked property. If a node contains *never* $P$, it cannot contain formulas of that type.

- If the node is a *send* event, then it contains a maximal noncontradicting subset of the formulas of the form $\{match\}\varphi$, where $\varphi$ is a subformula of the checked property. Otherwise, it cannot contain formulas of that type.
- Exactly one process predicate $P$, for one of the processes of the checked HMSC, except for the first node, which contains all the process predicates.
- If a node contains the process predicate $P$, then for each formula of type $\langle P\rangle\varphi$ that it contains, it also contains $\varphi$.

The liveness component is responsible, together with an appropriate acceptance condition that will described later, to guarantee the following: in every accepted run of the automaton, for all the nodes where a subformula of the form $\varphi U\psi$ hold, $\psi$ will hold in the current or a subsequent node. It contain a subset of the formulas of the form $\varphi U\psi$ and $\langle P\rangle\varphi U\psi$ that were mentioned above.

We have a set of directed edges between pairs of nodes such that the following constraints are satisfied. The constraints are described according to the type of subformulas that appears in the safety component of a given node. They apply to each node that contains such a formula and to its immediate successor according to the relation $\delta^A$. When an enumerated list of constraints is given for a certain type of subformula, then at least *one* of them must apply. The condition on the safety part appears in roman fonts, while the condition on the liveness component is given in *italics*. In the construction, we denote the process predicate in the current node by $Q$.

- $\exists\bigcirc\varphi$. (1) $\langle Q\rangle\varphi$ holds for each successor, or (2) the current node has both the proposition *send*, and $\{match\}\varphi$.
- $\forall\bigcirc\varphi$. (1) The current node has *receive* or *local*. Each successor has $\langle Q\rangle\varphi$, or (2) the current node has *send* and $\{match\}\varphi$. Each successor has $\langle Q\rangle\varphi$, or (3) the current node has *send* and $\{match\}\varphi$. Each successor has NEVER $Q$.
- $\varphi U\psi$. (1) The current node has $\psi$, or (2) the current node has $\varphi$, *send* and $\{match\}\varphi U\psi$, or (3) the current node has $\varphi$. Each successor has $\langle Q\rangle\varphi U\psi$. *If the liveness component of the current node contains $\varphi U\psi$, then the liveness component of each successor must have $\langle Q\rangle\varphi U\psi$.*
- $\varphi R\psi$. (1) The current node has both $\varphi$ and $\psi$, or (2) the current node has $\psi$, *send* and $\{match\}\varphi R\psi$, each successor has $\langle Q\rangle\varphi R\psi$, or (3) the current node has $\psi$, *send* and $\{match\}\varphi R\psi$, each successor has NEVER $Q$, or (4) the current node has $\psi$ and either *receive* or *local*. Each successor has $\langle Q\rangle\varphi R\psi$.

- $\varphi \vee \psi$. (1) The current node has $\varphi$, or (2) the current node has $\psi$.
- $\varphi \wedge \psi$. The current node has both $\varphi$ and $\psi$.
- $\langle P \rangle \varphi$. (1) The current node has $Q = P$, and $\varphi$. *If $\varphi$ is of the form $\delta U \psi$ and the liveness component has $\langle P \rangle \delta U \psi$, then the liveness component of the current node must also have $\delta U \psi$*, or (2) the current node has $Q \neq P$. Each successor has $\langle P \rangle \varphi$, *If $\varphi$ is of the form $\delta U \psi$ and the liveness component has $\langle P \rangle \delta U \psi$, then the liveness component of each successor node must have $\langle P \rangle \delta U \psi$.*
- NEVER $P$. The current node has $Q \neq P$. Each successor has NEVER $P$.
- start. *If the liveness component of the current node is empty, then the liveness component of each successor consists of the formulas of the type $\varphi U \psi$ and $\langle P \rangle \varphi U \psi$ that are included in the safety component of the successor.*

The last case takes care of the situation where all the previous liveness constraints were satisfied, and new ones are loaded from the safety component. Synchronizing with nodes that start a linearization of an MSC, marked with the proposition start. guarantees that there are no forward edges that propagate liveness constraints 'over' the current node. A state $s$ is *forward compatible* with a state $t$ if the following conditions hold: (1) $s$ has a *send* proposition, (2) $t$ has a *receive* proposition, (3) for each formula of type $\{match\}\varphi$ in $s$, we have the formula $\varphi$ in $t$, and (4) *If the safety component of $s$ contains a formula of the form $\{match\}\delta U \psi$, then the liveness component of $t$ must contain $\delta U \psi$.*

Forward compatibility means here that we *allow* a forward edge between the nodes, rather than *enforce* it. We can define the nodes of $S^A$ of the constructed specification automaton by spliting the nodes defined above as follows: for a compatible pair of nodes $s$ and $t$, we split $s$ into $s_1$ and $s_2$, such that both copies have the same incoming edges as the original $s$. Furthermore, we set $C^A(s_1, t)$ but not $C^A(s_2, t)$. However, this may cause an explosion in the number of states. We take an alternative approach: we set up $C^A(s, t)$, but redefine the relation $C'$ of the intersection automaton as follows:

$$\{((s, s'), (t, t')) | (s, s') \in S' \wedge (t, t') \in S' \wedge (s, t) \in C^B \rightarrow (s', t') \in C^A\}$$

The asymmetric definition of the intersection is due to the fact that the set $S^A$ is a compact representation of the actual states of the property automaton $A$, as described above.

We also have to take care that formulas that need to be satisfied further down the linearization will eventually be satisfied, rather than

be postponed forever. This is done using the Büchi acceptance set: it contains all the states that include the empty liveness component. This guarantees that we satisfy all the constraints that were loaded at the previous node with start. New such constraints will be loaded at the next such node. Notice that unlike the LTL construction in [11], the liveness constraints here may change, e.g., from $\langle P \rangle \eta U \mu$ into $\eta U \mu$.

The automaton $B$ recognizes a single linearization for each execution of a given HMSC $N = (\mathcal{M}, \tau, M_0)$. Take any MSC $M \in \mathcal{M}$. It is simple to construct an automaton $A_M$ that recognizes a single linearization of $M$. The $i$th node along the single execution of this automaton is labeled with the propositions of the $i$th event in the linearization. The linearization can be done using a topological sort in linear time in the number of events of $M$. The first node in each such linearization is marked with start.

Denote by $\mathit{first}(A_M)$ the initial state of $A_M$ and by $\mathit{last}(A_M)$ the final state. We would like to combine the different automata $A_M$ for $M \in \mathcal{M}$. Let $M^1, M^2, \ldots, M^k$ be the successors of $M$ under the relation $\tau$. That is $M \tau M^i$ for $1 \leq i \leq k$. To that, we add the set of transitions $(\mathit{last}(A_M), \alpha, \mathit{init}(A_{M_i}))$ for $1 \leq i \leq k$. That is, after executing $A_M$, there is a nondeterministic choice to any of the initial states of the automata representing the successor MSCs of $M$. Label these edges with the proposition start. The implementation automaton is linear in the size of the checked HMSC. Such an automaton for the HMSC that appears in Figure 2 is shown on Figure 3. Forward edges are denoted with dotted arrows. Nodes that are marked with an asterisk '$*$' are labeled with start.

## 5.1. COMBINING THE TWO AUTOMATA

Emptiness of automata with two successor relations is in general undecidable [6]. However, we obtain here decidability, and in fact, a decision procedure that is similar to the one for traditional Büchi automata through of the following restrictions: One of the relations (the forward edges) is embedded in the transitive closure of the other relation (representing the linearization successors). There is a finite bound on the number of successor edges between the source and the target of each forward edge. This is due to the fact that communication is limited to occur within a single HMSC node. Furthermore, there are no forward edges (i.e., the second relation) that 'overtake' the accepting states, i.e., start before an accepting node and ending after it.

Due to the the forward edges, the intersection of the two automata has an additional constraint over the construction in Section 4. During

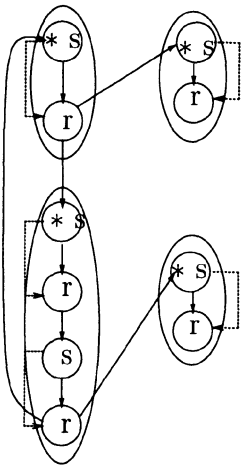


*Figure 3* An implementation automaton for the HMSC in Figure 2

the DFS performed to check the emptiness of the intersection, when we reach a *receive* node, we trace back the corresponding *send*. It must be already on the DFS stack. In fact, we can consult the implementation automaton $B$, which fixes the distance of the *send* event corresponding to the current *receive* event. In the intersection DFS stack, the *send* node would be down the stack the same distance from the *receive* node as it is in $B$. We then check the components of the property automaton $A$ on the DFS stack for forward compatibility. If there is no compatibility, we backtrack. Due to the above mentioned constraints on the forward edges relation, checking for emptiness of the intersection is done by simply looking for a reachable cycle through an accepting state of the intersection. The above property translation gives a Büchi automaton $A$ whose size may be exponential in the size of the checked formula and the number of processes. The implementation automaton $B$ is linear in the size of the checked HMSC.

## 6. DISCUSSION

Selecting a specification formalism often involves tradeoffs. A more concise specification often involves a higher complexity decision procedure. Adding expressiveness can also increase the complexity of even result in undecidability. In the case of HMSCs, the unboundedness of the message queues renders traditional specification formalisms such as LTL undecidable. One way to fight this problem is by imposing constraint on the HMSCs, as done e.g., in [2, 9].

In this paper, an alternative solution was suggested. The specification for HMSCs is based on partial order semantics rather than on interleaving semantics. The logic $\text{TLC}^-$ is only one candidate for such a specification formalism. Other logics such as $\text{ISTL}^F$ [4] can also be used.

Executions of HMSCs are partial order structures that do not have a fixed dependency relation between events according to the processes in which they participate. For example, if we have two sends from process $P1$ to process $P2$, we can have the linearizations $(s, P1)$ $(r, P2)$ $(s, P1)$ $(r, P2)$ and $(s, P1)$ $(s, P1)$ $(r, P2)$ $(r, P2)$, but cannot commute between the *send* and *receive* events on these sequences further. Thus, we *cannot* freely commute *send* and *receive* events of different processes. A previous construction for the full logic TLC [1] relied on a fixed dependency relation between events. The construction in the current paper, with forward edges and forward compatibility checks, is appropriate for FIFO asynchronous message passing, as in the visual semantics of HMSCs.

Since MSCs are used in very early stages of the design, verifying them can help in achieving early fault detection. Several logics for partial order executions were suggested in the last decade [4]. However, the linear semantics approach is still by far the more common way for specifying concurrent systems. For systems of MSCs, the partial order based model checking approach presented here seems natural and efficient. This may promote further research into such specification formalisms, their expressiveness and complexity.

A prototype implementation of HMSC model checking was done in SML/NJ. The HMSC was translated from an HMSC notation where each individual MSC is described using the ITU Z120 standard. Code that was used for HMSC pattern matching [10] was reused for implementing the translation of the HMSCs and the linearizations. The translation results in PROMELA code, which is the input language for the SPIN model checking system [8]. After translating an HMSC and a $\text{TLC}^-$ formula into PROMELA code, the SPIN model checking tool can be used to make the automatic verification. If the HMSC does not satisfy the specification, a counterexample is found. Simulating the counterexample using SPIN prints out a sequence of the HMSCs nodes that participate in the counterexample. Then, one can use the POGA tool to display the MSC execution.

## References

[1] Alur R., Peled D., and W. Penczek (1995), Model-Checking of Causality Properties, LICS'95, 10th Symposium on Logic in Com-

puter Science, IEEE, 1995, 90–100,

[2] Alur R., and Yannakakis M. (1999), Model Checking of Message Sequence Charts, CONCUR'99, Concurrency Theory, LNCS 1664, Eindhoven, Holland, 1999.

[3] Alur R., Holzmann G.J., and Peled D. (1996), An Analyzer for Message Sequence Charts, Software Concepts and Tools, Vol. 17, No. 2, pp. 70–77, 1996.

[4] Alur R. McMillan K., and Peled D. (1998), Deciding global partial-order properties, ICALP'98, Aalborg, Denmark, July, 1998, LNCS 1443, Springer, 41-52.

[5] Clarke E.M., and Emerson E.A. (1981), Design and synthesis of synchronization skeletons using branching time temporal logic. Workshop on Logic of Programs, Yorktown Heights, NY, 1981, LNCS 131, Springer-Verlag.

[6] Giammarresi D., Restivo A., Seibert S., and Thomas W. (1993), Monadic second-order logic over rectangular pictures and recognizability by tiling systems, Information and Computation 125 (1996), 32–45.

[7] ITU (1993), ITU-T Recommendation Z.120, Message Sequence Chart (MSC), March 1993.

[8] Holzmann G.J. (1992), *Design and Validation of Computer Protocols*, Prentice Hall Software Series, 1992.

[9] Muscholl A., and Peled D. (1999), Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces, Mathematical Foundations of Computer Science, Szklarska Poreba, Poland, September 1999, LNCS 1672, Springer, 81–91.

[10] Muscholl A., Peled D., and Su Z. (1998), Deciding Properties for Message Sequence Charts, FoSSaCS 1998, Foundations of Software Science and Computation Structures, Lisbon, Portugal, LNCS 1378, 226-242.

[11] Vardi M.Y., Wolper P. (1986), An automata-theoretic approach to automatic program verification. Proc. 1st Annual Symposium on Logic in Computer Science IEEE, 1986, 332-344.