Analyzing Message Sequence Charts

Anca Muscholl
LIAFA, Université Paris 7
2, pl. Jussieu, case 7014
F-75251 Paris

Doron Peled
Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974, USA

June 14, 2000

Abstract

The ISO standard for MSC provides a useful tool for visualizing communication protocols. MSCs present a model for concurrency that is different from the model of finite state systems, used frequently in automated verification. Thus, the MSC model poses new and interesting problems related to automatic verification of communication protocols. In this paper, some of the recent results related to MSCs are surveyed.

1 Introduction

It is commonly agreed that catching software bugs is a highly important task. On the other hand, people that develop formal methods techniques and tools often find it very hard to transfer their technology into the software development industry. One of the reasons of this aparent contradiction is that formal methods research often result in formalisms that are different than the ones already used by the software development. Some of the formalisms, such as temporal logic or the Z notation, have moderate success among users of formal methods. However, the bulk of software specification is still done informally, while the most commonly used method for software reliability is testing.

One partial solution for this situation is the use of visual formalisms. These are supposed to be more intuitive than textual formalisms, in the sense that "a picture is worth a thousand words". Describing software in different ways, from different perspectives, and by emphasizing different aspects, can certainly lead to a better understanding, and perhaps to some new insight. This is also the motivation in the Unified Modeling Language, UML.

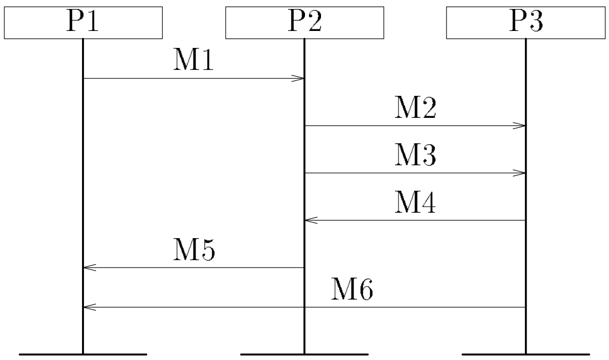

Figure 1: Visual representation of an MSC

Another group of formalisms is the ISO standards, including SDL, LOTOS and ESTELLE. These formalisms have both a textual as well as a visual representation. We will focus here on the ISO message sequence chart (MSC) notation. This formalism is often used in the design of communication protocols. A similar formalism is used in UML by describing interactions between objects.

The MSC formalism uses the partial order semantics instead of the usual interleaving semantics. As such, it has different properties than the more common models for concurrency. This calls for new algorithms and interesting decision problems. In this paper we will review some of the new results about MSCs.

2 Preliminary

Each MSC describes a scenario or an execution, 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 2.1 *An MSC M is a tuple $\langle V, <, \mathcal{P}, \mathcal{N}, L, T, N, m\rangle$.*

- *V is a (finite) set of events,*
- *$< \subseteq V \times V$ is a relation such that its transitive and reflexive closure $<^*$ is a partial order on V,*

- $\mathcal{P}$ *is a set of processes,*
- $\mathcal{N}$ *is a set of* message names*,*
- $L : V \longrightarrow \mathcal{P}$ *is a mapping that associates each event with a process,*
- $T : V \longrightarrow \{\mathrm{s}, \mathrm{r}, \mathrm{l}\}$ *is a mapping that describes each event as send,* receive *or* local*.*
- $N : V \longrightarrow \mathcal{N}$ *maps every event to a name.*
- $m : V \longrightarrow V$ *is a partial function called the* matching *that pairs up send and receive events. Each send $s \in V$ is paired up with exactly one receive $r \in V$ and vice versa, which is denoted $m(s) = r$. Events e and f can be paired up with each other, only if $N(e) = N(f)$.*

A message consists of a pair of matching send and receive events. For two events e and f, we have $e < f$ if and only if one of the following holds:

- *e and f are a matching send and receive events, respectively.*
- *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. Then, the following condition must hold:

$$\begin{aligned}&(T(e_1) = T(e_2) = \mathrm{s} \wedge T(f_1) = T(f_2) = \mathrm{r}\\&\wedge m(e_1) = f_1 \wedge m(e_2) = f_2 \wedge L(e_1) = L(e_2) \wedge\\&L(f_1) = L(f_2) \wedge e_1 < e_2) \rightarrow f_1 < f_2\end{aligned}$$

A *type* is an ordered pair of processes. Each send or receive event has a type, according to the origin and destination of the message. Matching events have the same type.

We call the transitive and reflexive closure $<^*$ of the relation $<$ the *visual ordering* of events. The order $<$ can be obtained from the syntactical representation of the chart (e.g. represented according to the standard syntax ITU-Z120).

The partial order between the send and receive events of Figure 1 is shown in Figure 2. In this figure, only the 'immediate precedes' order is shown. The MSC in Figure 1 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

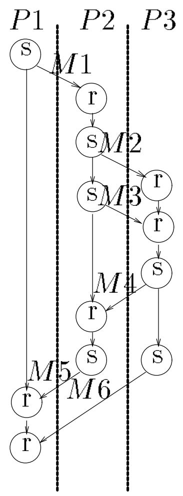

Figure 2: The partial order between the events of the MSC in Figure 1

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.

Definition 2.2 *The* concatenation *of two MSCs* $M_1 = \langle V_1, <_1, \mathcal{P}, \mathcal{N}_1, L_1, T_1, N_1, m_1 \rangle$ *and* $M_2 = \langle V_2, <_2, \mathcal{P}, \mathcal{N}_2, L_2, T_2, N_2, m_2 \rangle$ *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 the MSC* $\langle V_1 \cup V_2, <, \mathcal{P}, L_1 \cup L_2, T_1 \cup T_2, N_1 \cup N_2, m_1 \cup m_2 \rangle$*, where*

$$< \, = \, <_1 \cup <_2 \cup \{(e, f) \mid L_1(e) = L_2(f),\ e \in V_1 \text{ and } f \in V_2\}$$

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 different node. We can use the concatenation repeatedly, and define an infinite concatenation in an analogous way.

Since a communication system usually includes many such scenarios, a high level description is needed for combining them together. The standard description consists of a graph called HMSC (high level MSC), where each node contains one MSC as in Figure 3. 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*

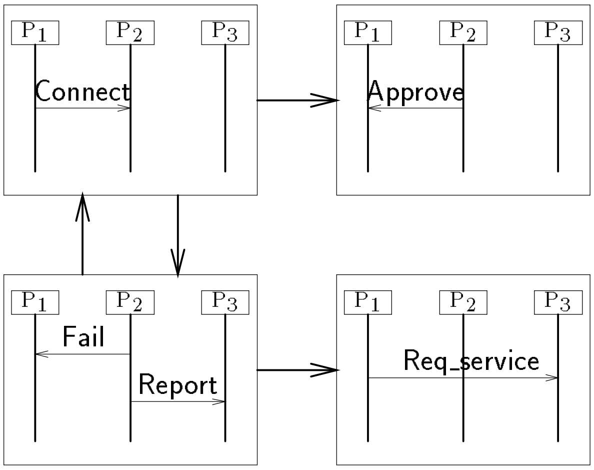

Figure 3: An HMSC graph

or *scenario*. Such an execution can be used to denote the communication structure of a typical ('sunny day') or an exceptional ('rainy day') behavior of a system, or a counterexample found during testing or model checking.

Definition 2.3 *An HMSC N is a triple $\langle \mathcal{S}, \tau, s_0, \lambda \rangle$ where $\mathcal{S}$ is a finite set of states. Each state s is labeled by some finite MSC $\lambda(s)$, all MSCs having the same set of processes, with sets of events disjoint from one another. $\tau \subseteq \mathcal{S} \times \mathcal{S}$ is the* edge relation *and the* initial *state is $s_0 \in \mathcal{S}$. An* execution *of N is a path $\xi = s_0\, s_1\, s_2\, \ldots$ of N that starts with the initial state s_0 and either ends with a state without outgoing edges, or is infinite.*

Figure 3 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 system are either finite or infinite. In Figure 3, process $P2$ may send its Report message after process $P1$ has progressed into the next node and has sent its Req_service message.

3 Undecidability Results

Since HMSCs do not put any constraint on the message queues, they can represent infinite state systems. For completeness, we will reprove known results [3, 8] about the undecidability of certain model checking problems for HMSCs. This will be done using a reduction from Post Correspondence Problem (PCP). An instance of PCP is a set of pairs of words

$$C = \{(v_1, w_1),\ (v_2, w_2),\ \ldots,\ (v_m, w_m)\}$$

over the alphabet $\Sigma = \{a, b\}$. We want to find out if there is a non-empty sequence of indexes $i_1, i_2, \ldots i_n$ such that $v_{i_1} v_{i_2} \ldots v_{i_n} = w_{i_1} w_{i_2} \ldots w_{i_n}$.

We will construct HMSCs with six processes P_1 to P_6 as follows:

- A message from P_1 to P_2 corresponds to the letter $a \in \Sigma$, a message from P_2 to P_1 corresponds to b. For each left word v_i in a PCP pair, there is a state labeled by an MSC M_i, where the messages sent between P_1 and P_2 appear according to the order of a, b in v_i.

- A message from P_3 to P_4 corresponds to a, and a message from P_4 to P_3 stands for b. For each right word w_i in the PCP pairs, there is a state labeled by an $M_i{}'$, where messages sent between P_3 and P_4 appear according to the order of a, b in w_i.

We will first show that the problem of emptiness of the intersection of two HMSCs is undecidable. This problem can be seen as a natural generalization of automata theoretical model checking [11] to the framework of HMSCs. In that problem, both the verified system and the (negation of the) checked property are represented using automata. If their intersection is nonempty, it must contain a counterexample for the checked property.

The first HMSC M has the following structure: s_0 is the initial node, labeled by a message from P_5 to P_6. There is a loop $s_0 \to M_i \to M_i{}' \to s_0$ for each pair of words $(v_i, w_i) \in C$. There is also an edge from $M_i{}'$ to a sink state s_1 and a self loop around s_1. The state s_1 is labeled by the empty MSC. The second HMSC N has the initial node s_0', which is labeled by the empty MSC. In addition, there is a node labeled by the MSC M_a, and one labeled by the MSC M_b. The MSC M_a contains one message from P_1 to P_2 and one message from P_3 to P_4, corresponding to the letter a. Similarly, M_b contains one message from P_2 to P_1 and one message from P_4 to P_3. The HMSC N contains a loop $s_0' \to M_\alpha \to s_0'$ for each $\alpha \in \{a, b\}$, and an edge from M_α to s_1'. The state s_1' is labeled by a message from P_5 to P_6 and has a self loop.

Messages from P_5 to P_6 appear only in s_0 and s_1'. This guarantees that after a finite number of iterations, a match must reach the nodes s_1 and s_1', giving a finite PCP match.

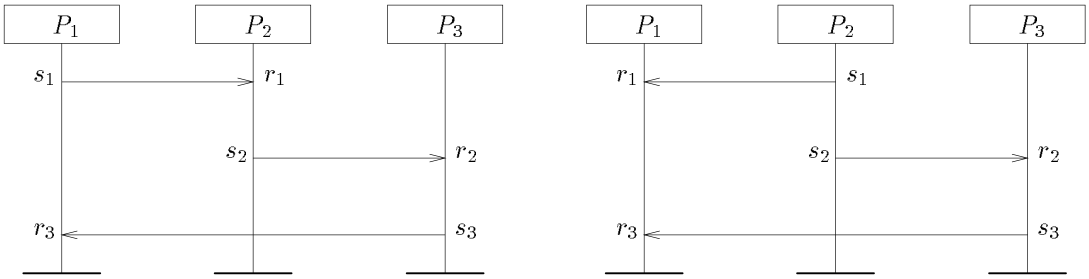

Figure 4: Two simple MSCs

Thus, the HMSC M is responsible for having pairs of words from C. Since the v_i words and the w_i words appear as sequences of messages between disjoint pairs of processes, the fact that M_i, representing v_i, and M_i', representing w_i appear in that order does not matter. The HMSC N is responsible for the matching between the two words, letter for letter. Each execution in the intersection of the two HMSCs correspond to a PCP match. The same proof idea works also for LTL model checking, since we can simply replace the HMSC N with an appropriate LTL formula. That formula asserts that in every execution a message from P_1 to P_2 is followed by one from P_3 to P_4, and conversely, etc.

4 Detecting Races

One of the simplest and most basic decision problems related to MSCs is to find *race conditions* [1]. To first see the intuition behind this problem, consider the two MSCs in Figure 4. In the left MSC, it is reasonable to infer that receive event r_3 occurs after the send event s_1. The intuition is that P_2's send event s_2 is delayed until the arrival of r_1, and P_3's send event s_3 is delayed until the arrival of r_2. Since a message cannot be received before it is sent, we have

$$s_1 < r_1 < s_2 < r_2 < s_3 < r_3.$$

However, it is not clear if the receive event r_1 precedes the receive event r_3 in the right MSC in Figure 4. It is possible that the message sent from P_2 to P_1 takes longer than the total time it takes for the messages from P_2 to P_3 and then from P_3 to P_1. Although the user may be led to assume, based on the visual order, that r_3 must always follow r_1, this is not necessarily the case. An implementation of the protocol that is based on this assumption may encounter unspecified reception errors, it may deadlock, or, end up deriving information from the wrong message.

We define the *enforced* order $\ll$ to contain all the event pairs that the underlying architecture can guarantee to occur only in the order specified. For example, if a send event s follows a receive event r in the visual order, then the implementation can force the process to wait for the receive event r before allowing the send event s to take place. The message sent may, for instance, need to carry information that is acquired from the received message r. On the other hand, the architecture may fail to force two receives that arrive from different processes to appear according to the visual order.

The enforced order can be obtained from a given MSC using a set of conditions that are architecture-dependent. The order $\ll$ contains exactly the pairs (e, f) of events that satisfy at least one of the conditions. As an example, consider the following conditions for obtaining the enforced order for fifo semantics:

Two sends from the same process: $T(e) = T(f) = s$, $L(e) = L(f)$, and $e < f$.

A message pair: $T(e) = s$, $T(f) = r$ and $m(e) = f$.

Messages ordered by the fifo queue: $T(e) = T(f) = r$, $L(e) = L(f)$, $e < f$ and $\exists e' \exists f' \, (m(e) = e', \, m(f) = f', \, L(e') = L(f')$ and $e' < f')$.

A receive precedes a send on the same process line: $T(e) = r$, $T(f) = s$, $L(e) = L(f)$ and $e < f$.

Since the enforced order $\ll$ corresponds to the causality in the system, one can compute the order $\ll^*$ among the set of events, i.e., its reflexive and transitive closure. It can be calculated using the Floyd-Warshall algorithms. It can then be checked whether $<$ is a subset of $\ll^*$.

Definition 4.1 *A race occurs in an MSC if it is not the case that* $< \subseteq \ll^*$.

For example, the race in the right MSC in Figure 4 corresponds to the interpretation where $\ll$ includes

$$\{(s_1, r_1), (s_1, s_2), (s_2, r_2), (r_2, s_3), (s_3, r_3)\},$$

while (r_1, r_3) is not in $\ll^*$.

Unfortunately, when moving from simple MSCs, which represent single execution scenarios, to an HMSC, the situation is not that simple [7]. Since an HMSC represents a collection of scenarios, it is reasonable to allow races in single executions, under the requirement that every race occurring in some execution of the HMSC must be covered by some other execution.

Definition 4.2 *An HMSC M has a race if for at least one of its executions χ and one linearization w of the enforced order $\ll$ of χ there is no execution χ' of M such that w is the linearization of the visual order $<$ of χ'.*

Theorem 4.3 ([7]) *The race problem for HMSCs is undecidable.*

The following reduction from PCP will show the above result. We define an HMSC M over seven processes, $P_1, \ldots, P_7$. The given PCP instance is a set of pairs of words

$$C = \{(v_1, w_1), \, (v_2, w_2), \, \ldots, \, (v_m, w_m)\}$$

over the alphabet $\Sigma = \{a, b\}$. We encode the letter a by a message Ma from P_1 to P_2, and b by a message Mb from P_1 to P_3. In the same way, we encode indices of PCP pairs using processes P_4, P_5, P_6: the digit 0 (1, respectively) is coded by a message $M0$ from P_4 to P_5 ($M1$ from P_4 to P_6, respectively). Finally, messages $M2, M3, M5, M6$ are sent from P_7 to P_2, P_3, P_5, P_6, respectively. We assume that the only difference between $<$ and $\ll$ is that two receives on the same process where the sends are on different processes are not ordered by $\ll$. Thus, there can be only races between for example, $M2$ and Ma (or $M3$ and Mb, and so on).

We'll use for each u_i, $i \in \{1, \ldots, m\}$, two sorts of MSCs over $P_1, \ldots, P_6$. Let $\mathrm{Eqlen}_{i,v}$ be the set of all MSCs encoding the index i over P_4, P_5, P_6 and a word from $\{a, b\}^*$ of equal length as v_i over P_1, P_2, P_3. By $\mathrm{Neq}_{i,v}$ we denote the HMSC encoding all pairs (i, v) with $v \in \{a, b\}^*$ and $v \neq v_i$. We suppose that $\mathrm{Neq}_{i,v}$ has a unique sink (final) state. We define analogously $\mathrm{Eqlen}_{i,w}$ and $\mathrm{Neq}_{i,w}$.

The HMSC M has distinguished states $s_0, \ldots, s_3$ and consists of three parts. The first two parts are related to the words u_i, v_i, respectively. The initial state s_0 is labeled by the empty MSC and has arcs $s_0 \to s_1, s_0 \to s_2$ to the empty labeled states s_1, s_2. Around s_1 there are loops $s_1 \to E \to s_1$ for each $E \in \mathrm{Eqlen}_{i,v}$. Moreover, s_1 has an arc to the initial state of $\mathrm{Neq}_{j,v}$. For the final state f of $\mathrm{Neq}_{j,v}$ we have the following transitions: arcs from f to new states labeled by $E' \in \mathrm{Eqlen}_{k,v}$, for all k, and an arc from f to s_3. Around each state labeled by $E' \in \mathrm{Eqlen}_{k,v}$ there is a self-loop and an arc to s_3. The state s_3 is labeled by the MSC $M2, M3, M5, M6$. The same construction is done starting with s_2 for all words w_i.

The third component of M corresponds to all message sequences from $(Ma + Mb + M0 + M1 + M2 + M3 + M5 + M6)^*$ which contain

- at least one $M0$ or $M1$, and
- exactly once $M2$, $M3$, $M5$, $M6$ respectively, in this order, and
- not simultaneously $M2$ after all occurrences of Ma, $M3$ after all Mb, $M5$ after all $M0$ and $M6$ after all $M1$.

For each maximal finite path in the third component of M, messages $M2, M3, M5, M6$ can be postponed until all other messages have been sent and received. It can be checked that M is race-free if and only if every sequence from $\{a, b, 0, 1\}^*$ containing at least one 0 or 1 can occur in the first two components of M. In these components, finite paths (up to s_3) correspond exactly to pairs $(i_1 i_2 \cdots i_k, x)$, $k > 0$, where either $x \neq v_{i_1} \cdots v_{i_k}$ or $x \neq w_{i_1} \cdots w_{i_k}$. Thus, M is race-free if and only if the PCP instance has no solution.

5 Positive Model Checking Results

We saw in Section 3 that the emptiness of the intersection of two HMSCs is undecidable, and so is LTL model checking. This does not mean that we should abandon the popular MSC notation. One possible solution is to impose constraints on HMSCs. A simple constraint is to limit message queues to some predefined values. Of course, this not suffice, as one can see from the undecidability proof for model-checking, where we can even suppose that communication is synchronous. A stronger constraint appears e.g., in [7, 3]. In this section we will show alternative solutions: decision procedures that are specialized to the HMSC notation.

In the first solution, we denote the specification using MSCs that are called a *template*. A template denotes a set of events and their relative order. A template MSC has the same form as an MSC, and a template HMSC has the same form as a template HMSC. A template MSC *matches* any scenario MSC that contains at least those events that appear in the template, while preserving the *enforced* order between them. The matching path of the design can have additional events besides the ones appearing in the template. At the conclusion of the search, the template matching can provide a matching scenario, or the fact that no matching scenario exists in the checked hierarchical graph.

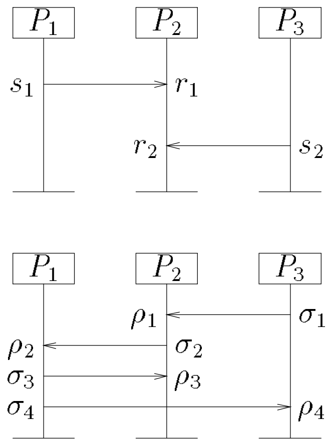

Figure 5: A template (top) and a matching scenario (bottom)

The match can be used for determining whether MSCs with unwanted properties exist in the design. Another use is for determining whether a required feature is already included in the design or remains to be added. An example of a template and a matching MSC scenario appears in Figure 5. In both charts, there are three processes, P_1, P_2 and P_3. The result of this match is that s_2 is paired up with σ_1, r_2 with ρ_1, s_1 with σ_3, and r_1 with ρ_3. Notice r_1 precedes r_2 in the visual order, while ρ_1 precedes ρ_3 in the visual order. However, the matching is done with respect to the enforced order, in which r_1 and r_2 are unordered, and so are ρ_1 and ρ_3.

We can define now template matching formally:

Definition 5.1 *Let O be a 'template' MSC, $\langle V_1, <_1, \mathcal{P}, \mathcal{N}, L_1, T_1, N_1, m_1\rangle$, and let M be an MSC $\langle V_2, <_2, \mathcal{P}, \mathcal{N}, L_2, T_2, N_2, m_2\rangle$. Let $\ll_1$ be the enforced order for O, and $\ll_2$ be the enforced order for M. We say that O* matches *M if there is an injective mapping $\Gamma : V' \mapsto V$ such that*

1. *$L_1(e) = L_2(\Gamma(e))$.*
2. *$T_1(e) = T_2(\Gamma(e))$.*
3. *$N_1(e) = N_2(\Gamma(e))$.*
4. *If $m_1(e) = f$ then $m_2(\Gamma(e)) = \Gamma(f)$.*
5. *If $e \ll_1 f$ then $\Gamma(e) \ll_2 \Gamma(f)$.*

Definition 5.2 *Let H_1 be an HMSC template, and H_2 and HMSC. Then H_1* matches *H_2 if there is a pair of executions, O of H_1 and M of H_2 such that O matches M.*

Theorem 5.3 ([8]) *Template matching for HMSCs is decidable and is an NP-complete problem.*

A different solution is to use for specification temporal logics that are interpreted over partial orders, such as TLC [2], or LTrL [12]. These logics are interpreted directly over HMSC executions, rather than interleaving sequences (as LTL). Thus, they do not distinguish between interleaved sequences that correspond to the same partial order execution (or in terms of MSCs, a scenario). This allows obtaining back the decidability of model checking [9].

6 HMSC Specifications

When using HMSCs for specifying communication protocols, the user has to keep in mind that each state is labeled by a complete chart, i.e., every send has to be matched by a receive. However, some simple finite state communication protocols cannot be modelized in this way. The example in Figure 6 appears in [4].

For a finite (Büchi) automaton $\mathcal{A}$ over an alphabet Σ let $L(\mathcal{A})$ be the set of finite and infinite words over Σ which are accepted by $\mathcal{A}$. For sake of simplicity, we omit in the following message names and local actions, however they can be easily included in our formalism. Let $\mathcal{P}$ be the (finite) set of processes and let $\Sigma = \{P?Q, P!Q \mid P, Q \in \mathcal{P}\}$ be a set of (communication) actions. For an HMSC M, $\mathrm{Lin}(M) \subseteq \Sigma^\omega$ is the set of sequences of (types of) events, which are linearizations of some execution of M.

Definition 6.1 *A transition system $\mathcal{A}$ over Σ is* HMSC-realizable*, if some HMSC M exists such that $L(\mathcal{A}) = Lin(M)$.*

Consider a finite state protocol, specified as a labeled transition system $\mathcal{A}$. In general, such a protocol cannot be rewritten into an HMSC. Hence, we might want to test whether $\mathcal{A}$ is MSC-realizable and if this is the case, compute an equivalent HMSC.

The set $L(\mathcal{A})$ should of course satisfy at least the following constraints. First, $L(\mathcal{A})$ has to be definable by a set of forbidden prefixes $F \subseteq \Sigma^*$. That is, a sequence v belongs to $L(\mathcal{A})$ if and only if it has no finite prefix in

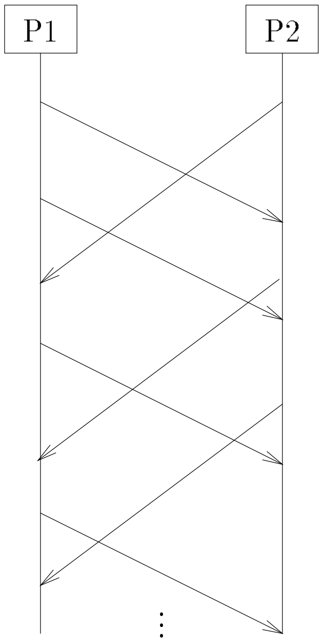

Figure 6: A non HMSC-realizable protocol.

F. This property is equivalent to requiring that $L(\mathcal{A})$ can be recognized by an automaton where every state is final. It is well-known that verifying this property is a PSPACE complete problem, see e.g. [5]. Second, $L(\mathcal{A})$ can consist of well-formed words only, i.e., every prefix u of some $v \in L(\mathcal{A})$ satisfies $|u|_a \geq |u|_{\bar{a}}$, for every pair $(a, \bar{a}) = (P!Q, Q?P)$. Of course, we cannot require for a pair $(a, \bar{a}) = (P!Q, Q?P)$ that every $v \in L(\mathcal{A})$ can be decomposed in $v = v_0 v_1 \ldots$ such that $|v_i|_a = |v_i|_{\bar{a}}$ for any i. Take for example the sequence $P!Q\, P!Q (Q?P\, P!Q)^\omega$, which is derived from the realizable sequence $(P!Q\, Q?P)^\omega$ by commuting the first two actions. We can test whether $L(\mathcal{A})$ contains only well-formed words in polynomial time, using for example Floyd's algorithm. Note that for any finite word u labeling a path in $\mathcal{A}$, the difference $|u|_a - |u|_{\bar{a}}$ is bounded by the number of states of $\mathcal{A}$, for every pair of matching events $(a, \bar{a}) = (P!Q, Q?P)$. Finally, $L(\mathcal{A})$ must be closed under the partial order $\leq$, that is, for any linearizations u, v of the same $\leq$ partial order, $u \in L(\mathcal{A})$ if and only if $v \in L(\mathcal{A})$. Testing this closure property is a PSPACE complete problem, too, [6, 10].

We propose below a criterion for testing the HMSC-realizability of a regular language $L \subseteq \Sigma^\infty$. A finite word u is called *balanced*, if it is well-formed and $|u|_a = |u|_{\bar{a}}$, for every pair $(a, \bar{a}) = (P!Q, Q?P)$. We define $\mathcal{F}_{\min}(L)$ as the set of all balanced factors u of sequences from L satisfying the following condition:

For every send event s which is not at the first position in u there is some pair of matching events in u, $m(e) = f$, such that $e < s < f$.

The previous condition means in particular that the partial order defined by some word $u \in \mathcal{F}_{\min}(L)$ cannot be decomposed into a product of at least two partial orders of balanced words. The idea is that factors belonging to $\mathcal{F}_{\min}$ have to be located *within* a chart associated with a state. In most cases, $\mathcal{F}_{\min}$ will consist of single messages. For example, for the language $L = \mathrm{Lin}(M)$ in Figure 3 the set $\mathcal{F}_{\min}$ consists of all messages occurring in M. As a further example for $\mathcal{F}_{\min}$, note that the sequence $P!Q\ P!Q\ Q?P\ Q?P$ cannot belong to $\mathcal{F}_{\min}(L)$. On the other hand,

$$P!Q\ P!Q\ P!R\ P!S\ S?P\ P!T\ T?P\ T!Q\ Q?T\ Q?P\ Q?P\ S!R\ R?S\ R?P$$

can belong to a set $\mathcal{F}_{\min}(L)$.

Proposition 6.2 *Let $L \subseteq \Sigma^\infty$ be regular. L is HMSC-realizable if and only if $\mathcal{F}_{\min}(L)$ is finite.*

For one direction of the proof assume that $L = \mathrm{Lin}(M)$, for some HMSC M. Then every factor $u \in \mathcal{F}_{\min}(L)$ belongs to a chart labelling some state of M. Hence, $\mathcal{F}_{\min}(L)$ is finite.

For the converse, let $\mathcal{F}_{\min}(L)$ be finite, with L given by an automaton $L(\mathcal{A})$ where all states are final. We define an HMSC with states of the form (q, u), where q is a state of $\mathcal{A}$ and $u \in \mathcal{F}_{\min}(L)$. The state (q, u) is labeled by the chart obtained from u. We have an edge $(q, u) \longrightarrow (q', u')$, whenever there is a path in $\mathcal{A}$ from q to q' and labeled by u. The initial states of M are the states where the first component is the initial state of $\mathcal{A}$. It is easy to see that $\mathrm{Lin}(M) \subseteq L$. Conversely, L is generated by $\mathcal{F}_{\min}(L)$. That is, every $v \in L$ can be obtained from a sequence $v_0 v_1 \ldots$ with $v_i \in \mathcal{F}_{\min}(L)$ for all i, by commuting adjacent actions which are causally independent. Since L is closed under the partial order $\leq$, we get $L \subseteq \mathrm{Lin}(M)$.

Given an automaton $\mathcal{A}$ for L of size n and $|\mathcal{P}| = m$, a (non-deterministic) automaton $\mathcal{B}$ for $\mathcal{F}_{\min}(L)$ has size exponential in n, m. The automaton $\mathcal{B}$ uses for each pair of processes a counter (up to n) for checking well-formedness. Furthermore, $\mathcal{B}$ guesses for each intermediate send event s the (currently unmatched) send/receive pair $m(e) = f$ with $e < s < f$. Thus, if $L = L(\mathcal{A})$ is HMSC-realizable, then an equivalent exponential size HMSC M can be constructed.

References

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

[2] R. Alur, D. Peled, and W. Penczek. Model-checking of causality properties. In *Proc. of LICS '95*, pages 90–100. IEEE, 1995.

[3] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. of CONCUR'99*, LNCS 1664, Springer, 1999.

[4] E. Gunter and D. Peled. Compositional message sequence charts, 2000. Manuscript.

[5] O. Kupferman and M. Y. Vardi. Model checking of safety properties. In *Proc. of CAV'99*, pages 172–183, 1999.

[6] A. Muscholl. *Über die Erkennbarkeit unendlicher Spuren.* PhD thesis, Universität Stuttgart, 1994.

[7] A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proc. of MFCS'99*, LNCS 1672, Springer, 1999.

[8] A. Muscholl, D. Peled, and Z. Su. Deciding properties of message sequence charts. In *Proc. of FoSSaCS'98*, LNCS 1378, Springer, 1998.

[9] D. Peled. Specification and verification of message sequence charts, 2000. Submitted.

[10] D. A. Peled, T. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages. *Theoretical Computer Science*, 195(2):183–203, 1998.

[11] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. of LICS'86*, pages 322–331, 1986.

[12] I. Walukiewicz. Difficult configurations – on the complexity of LTrL. In *Proc. ICALP'98*, LNCS 1443, Springer, 1998.