Verification of Message Sequence Charts via Template Matching

Vladimir Levin and Doron Peled
Bell Laboratories
Lucent Technologies
700 Mountain Avenue
Murray Hill, NJ 07974

Abstract. Message sequence charts are becoming a popular low-level design tool for communication systems. When applied to systems of non-trivial size, organizing and manipulating them become a challenge. We present a methodology for specifying and verifying message sequence charts. Specification is given using *templates*, namely charts with only partial information about the participating events and their interrelated order. Verification is done by a search whose aim is to match templates against charts. The result of such a search either reports that no matching chart exists, or returns examples of charts that satisfy the constraints that appear in such a template. We describe the algorithm and an implementation.

1 Introduction

Message sequence charts are becoming more and more popular in the design of communication systems [5]. They allow a low level description of features the designed system ought to have. Description of a system via message sequence charts refers to *scenarios* of executions. An MSC specification contains usually a description of some typical executions of the system (sometimes called *sunny day scenarios*), and also some particular unusual executions (sometimes called *rainy day scenarios*) to which the system developer must pay extra attention.

The simplicity of the MSC model stem from its simple graphical representation, and from the correspondence between one MSC and a single execution of the designed system. However, to be useful, various groupings of scenarios need to be considered. When specifying a system of non-trivial size, organizing the different scenarios in a useful way becomes a problem. Another reason for grouping scenarios is that typically many scenarios reflect very similar executions, motivating the need to combine scenarios from smaller building blocks.

In this paper we suggest a methodology, an algorithm and a tool for organizing and manipulating families of MSC scenarios. We suggest a notation for describing a system of message sequence charts, which allows expressing concatenation and alternation between charts. Then, we introduce the notion of

an *MSC template*, which allows denoting a partially specified execution. Such a template can be conceived as a specification of a desired or a forbidden feature, and can be checked against a system of MSCs. We show an algorithm for checking whether a template matches against a system of MSC scenarios. We discuss an implementation of the algorithm using the COSPAN [7] verifyer.

Our MSCs template search can serve for various purposes:

System validation. The template represents a specification of the system. It describes it in the 'negative', in the sense that no legal execution of the system can match the specification. If during a search a match is made, the specification does not hold for the system. The charts that match, and hence violate the specification, are detected and need to be re-examined.

Features update. The template is used to keep track of provided charts and features. A template represents a chart or a feature that needs to be represented. During updating of the MSCs, one can search the existing library of MSCs to check whether a chart that covers the case described by a given template already exists.

Creating system views. With a considerably big system, containing many charts, it is important to be able to provide the capability of observing different 'views' of the system. One way to obtain views is by using database queries. For example, viewing only the charts that contain a certain phrase in their title. Using template search, one can generate views that correspond to the *semantic* contents of the charts. Namely, displaying all charts that contain a certain interaction between the processes.

2 Charts and Templates

2.1 The syntax and semantics of message sequence charts

Let R^* be the transitive closure of a binary relation R. Let $\circ$ be the relation composition symbol. A relation R is called *reduced* if $(R \circ R \circ R^*) \cap R = \phi$, i.e., if there is a sequence $e_1 R e_2 R \ldots R e_n$ with $n > 2$, then $(e_1, e_n) \notin R$. R is *cycle free* if $R \circ R^*$ is nonreflexive.

Syntax: MSC scenarios MSC diagrams are graphical representations of scenarios or executions of communication systems. The representation is formally defined in [5]. Examples of MSC diagrams appear in Figures 1, and 2.

An MSC $\mathcal{M}$ is a fivetuple $\langle E, <, L, T, \mathcal{P} \rangle$, where E is a set of *events*, $< \subseteq E \times E$ is a cycle free relation, $\mathcal{P}$ is a set of *processes*, $L : E \mapsto \mathcal{P}$ is a mapping that assigns each event with a process, and $T : E \mapsto \{\mathsf{s}, \mathsf{r}\}$ maps each event to its type, i.e., *send* or *receive*.

The relation $<$ is called the *visual order* between events. It reflects the relative appearance of events in a graphical representation of the MSC. Thus, $e < f$ if either

- e and f are the send and receive events, respectively, of the same message, in this case, the events e and f are said to be a *matching pair.*
- e and f belong to the same process, with e appearing above f in the process line.

Let $E_{P_i} = \{e | e \in E \wedge L(e) = P_i\}$. Denote the *local visual order of process* P_i by $<_{P_i} = < \cap (E_{P_i} \times E_{P_i})$, and the *communication visual order* between sends and receives by $<_c = \{(e, e') | e < e' \wedge L(e) \neq L(e')\}$. Thus, $<= <_c \cup \bigcup_{P_i \in \mathcal{P}} <_{P_i}$.

Consider the MSC of Figure 1. We have $E = \{s_1, r_1, s_2, r_2\}$, $\mathcal{P} = \{P_1, P_2, P_3\}$, $<_c = \{(s_1, r_1), (s_2, r_2)\}$, $<_{P_1} = <_{P_3} = \phi$, and $<_{P_2} = \{(r_1, r_2)\}$. The visual order $<$ is depicted on the lower left side of the figure. This order is termed 'visual', since it reflects the way the MSC is depicted, but may differ from the actual execution order between events as explained below.

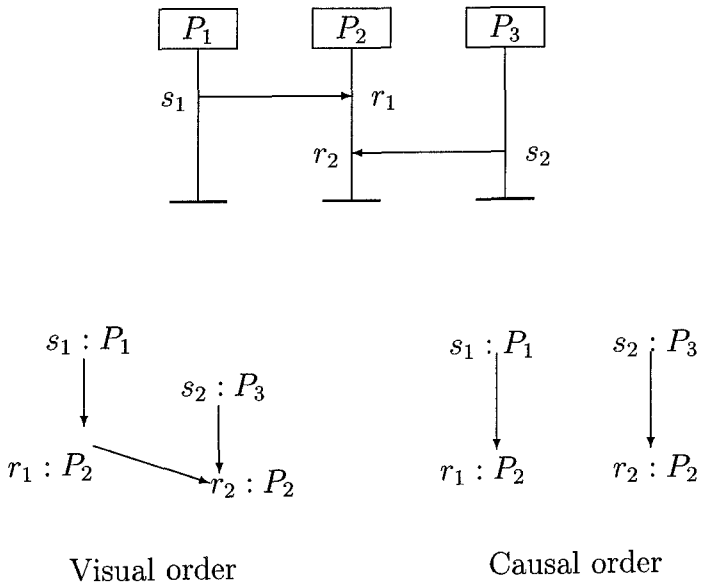

Fig. 1. A chart, its visual and precedence order

Semantics: Causal Structures Causal structures, akin to *pomsets* [11], *event structures* [10] and *traces* [9], are obtained from a message sequence charts and a selected semantics [1]. It represents one possible execution of a communication system. It contains information about the executed events, and the precedence order between them.

A causal structure $\mathcal{O}$ is represented by a fivetuple $\langle E, \prec, L, T, \mathcal{P} \rangle$, where the only component that differs here from the definition of an MSC is the cycle free

relation $\prec$. This relation is called the *precedence order*. That is, if $e_1 \prec e_2$, event e_1 must have terminated before event e_2 started. The transitive closure $\prec^*$ of $\prec$ is a partial order called also the *causal order*. Notice that two events that are unordered by $\prec^*$ can occur independently or concurrently with each other.

Considering again the MSC of Figure 1, the 'precedence' order, which appears on the lower right of the figure, reflects the execution order. The distinction between the visual order and the precedence order often reflects the shortcomings of a two dimensional representation of the MSC. For example, in the example of Figure 1 it is arguable whether the receive event r_1 actually precedes r_2, as these messages were sent independently from different processes. Placing them in a particular order can merely stem from the fact that the MSC representation forces *some* arbitrary visual order, rather than an explicit intent to assert that they actually arrive at this particular order.

The translation between the visual order and the precedence order is done via *semantic rules* [1], which select which ordered pairs of the visual order pertain at the precedence order. For example, one such rule asserts that $<_c \subseteq \prec$. The arbitrariness of the choice of order between r_1 and r_2 discussed above is reflected by the *absence* of a rule such that if $e_1 < e_2$, $T(e_1) = T(e_2) = \mathsf{r}$, and $L(e_1) = L(e_2)$, then $e_1 \prec e_2$. Notice that the semantic rules depend on the system's architecture. In a system where each process has multiple asynchronous communication queues, one can impose an arbitrary order on independently received messages, reflecting the order of *reading* the messages rather than their physical order of *arrival*. In such a system, letting $r_1 \prec r_2$ may be meaningful.

We will assume a fixed set of semantic rules. The causal structure obtained from a given MSC N by applying these rules will be denoted by $\mathcal{O} = tr(N)$.

One set of semantic rules, for an architecture with *fifo* queues, such that each process has one fifo message queue for all the incoming messages, sets $e_1 \prec e_2$ in the following cases:

Two sends from the same process.

$$T(e_1) = \mathsf{s} \wedge T(e_2) = \mathsf{s} \wedge L(e_1) = L(e_2) \wedge e_1 < e_2$$

A matching pair of send and receive.

$$T(e_1) = \mathsf{s} \wedge T(e_2) = \mathsf{r} \wedge L(e_1) \neq L(e_2) \wedge e_1 < e_2$$

We will denote this condition by $msg(e_1, e_2)$.

Fifo order.

$$T(e_1) = \mathsf{r} \wedge T(e_2) = \mathsf{r} \wedge e_1 < e_2 \wedge L(e_1) = L(e_2) \wedge \exists f_1 \exists f_2 (msg(f_1, e_1) \wedge msg(f_2, e_2) \wedge L(f_1) = L(f_2) \wedge f_1 < f_2)$$

A receive and a later send at the same process.

$$T(e_1) = \mathsf{r} \wedge T(e_2) = \mathsf{s} \wedge L(e_1) = L(e_2) \wedge e_1 < e_2$$

For a non-fifo architecture, one needs to remove the third (fifo) rule.

Notice that both visual and precedence orders, are not necessarily transitive closed or reduced. This is important for the efficiency of the matching algorithm described in the sequel. Thus, in Figure 2, $s_1 \prec s_2$, $s_2 \prec s_3$ and $s_1 \prec s_3$ hold.

This merely reflects the fact that the local visual order is a total order for each process, hence is transitive closed. On the other hand, although $s_1 \prec s_2$ and $s_2 \prec r_2$, it does not hold that $s_1 \prec r_2$.

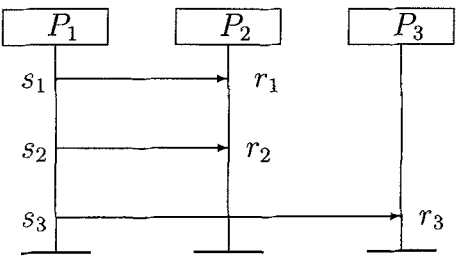

Fig. 2. Another MSC

2.2 A Calculus of Message Sequence Charts

An important feature of a system specification is compositionality: the ability to construct the description of a system from simpler and smaller building blocks. We first define the concatenation of MSCs.

Suppose we want to decompose the description of a chart into two tasks A and B, such that A occurs before B. We assume A and B agree on their sets of processes $\mathcal{P}_A = \mathcal{P}_B$. Denote the visual order of events in A by $<_A$, and in B by $<_B$. We define a *syntactic concatenation*. The events of each process in A appear before the events of the same process in B in the visual order. Thus, $<_{AB} = <_A \cup <_B \cup \{(e_1, e_2) | e_1 \in E_A \wedge e_2 \in E_B \wedge L(e_1) = L(e_2)\}$. The precedence order of the concatenation depends on the particular semantics chosen for the system. It is important to define that if the same MSC appears more than once in a concatenation, we use a disjoint set of events for each occurrence[1].

This concatenation is termed 'syntactic' since it behaves as if we drew the MSC B below the MSC A along the same process lines. It is related to the *layered decomposition* of concurrent systems [3, 6]. Denote the combination by AB, and accordingly, the precedence order of events by $\prec_{AB}$. The precedence order is obtained by applying the semantic rules to the above defined visual order $<_{AB}$. Thus, under our fifo queue semantics, when concatenating A with B in Figure 3, we have that r_4 and r_2 are not ordered according to $\prec_{AB}$.

Once the concatenation is defined, we allow combining charts using rational expressions. We allow the syntax

$$\mathcal{A} ::= B \mid (\mathcal{A}) \mid \mathcal{A}^* \mid \mathcal{A}\mathcal{A} \mid \mathcal{A} + \mathcal{A} \mid \varepsilon$$

[1] Technically, one can define the concatenation of A and B using two renamed sets of events: $E_A \times \{1\}$ and $E_B \times \{2\}$, with the order and the labeling functions relativized to the renamed sets of events.

with B denoting a variable representing an MSC.

The semantics of these rational expressions is as follows: The *empty MSC* ε contains no events. Let $\mathcal{A}$ and $\mathcal{B}$ range over *sets of MSCs*. Let $\mathcal{AB} = \{AB | A \in \mathcal{A} \wedge B \in \mathcal{B}\}$. Define $\mathcal{A}^0 = \varepsilon$, $\mathcal{A}^{i+1} = \mathcal{A}^i\mathcal{A}$. Then, $\mathcal{A}^* = \bigcup_{i=0}^{\infty} \mathcal{A}^i$. Finally, $\mathcal{A} + \mathcal{B} = \mathcal{A} \cup \mathcal{B}$.

Equivalently, we can specify a system of MSCs using finite graphs, with nodes corresponding to MSCs [8]. A finite path corresponds to an MSC obtained by syntactically concatenating the charts along it. The graph in Figure 3 corresponds to the rational expression $(AC)^*(\varepsilon + A + AB)$. Notice that each such rational expression, considered as a language, is prefix closed. The tool POGA supports storing and viewing graphs of MSCs [4].

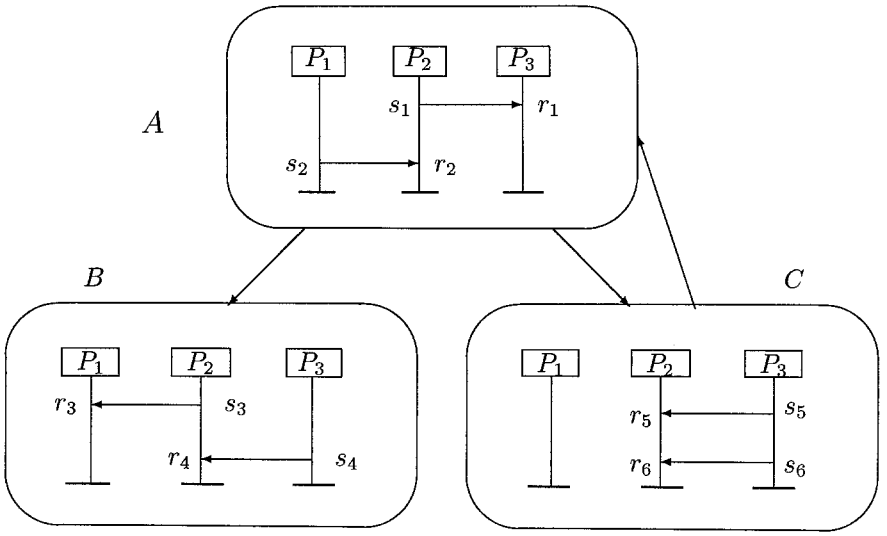

Fig. 3. A graph of MSCs

Recall that a *linearization* $\sqsubset$ of a precedence order $\prec$ is a total order that contains $\prec$. Notice that the language obtained by taking all the linearizations of an MSC rational expression may not necessarily correspond to a regular language. For example, consider the system described in Figure 4. It includes all the words (linearizations) with the same number of sends and receives such that any of their prefix contains no more receives than sends. This language is clearly not regular.

2.3 Templates

A *template* is also a chart. It has the same syntax as an MSC. Its semantics is similar to that of an MSC, except that unlike an MSC, the causal structure $tr(M)$

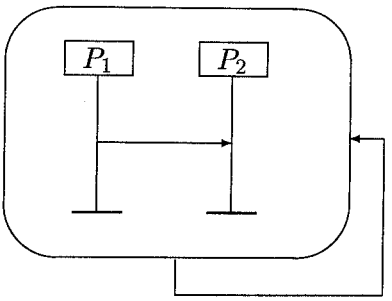

Fig. 4. An MSC system whose set of linearizations is not regular

corresponding to a template M contains an order relation $\prec_M$ that is *reduced*. This requirement follows a subtle efficiency argument that will be discussed in the sequel. Hence, if the chart in Figure 2 is interpreted as a template, we have $s_1 \prec s_2$ and $s_2 \prec s_3$, but $s_1 \not\prec s_3$. A template specifies an order between events. Conceptually, it does not correspond to a full scenario, but rather to a subset thereof. The lack of causal order (the transitive closure of $\prec_M$) between pairs of events in a template means that the order between the events is unimportant or unknown.

3 Correctness Criterion: Templates Matching

3.1 Matching a template against an MSC

A template *matches* or is *embedded* in an MSC, if the chart respects the order on the events specified by the template. Matching is defined with respect to a given semantics.

Definition 1 *Under a given semantics, a template M with a causal structure $tr(M) = \langle E_M, \prec_M, L_M, T_M, \mathcal{P}_M \rangle$ matches a chart N with a causal structure $tr(N) = \langle E_N, \prec_N, L_N, T_N, \mathcal{P}_N \rangle$ iff*

- *$\mathcal{P}_M \subseteq \mathcal{P}_N$, and*
- *there exists a homomorphism (called an* embedding*) $\mu : E_M \mapsto E_N$ such that*
  - *for each $e \in E_M$, $L_N(\mu(e)) = L_M(e)$ and $T_N(\mu(e)) = T_M(e)$ [preserving processes and types],*
  - *if $e_1 \prec_M e_2$, then $\mu(e_1) \prec_N \mu(e_2)$ [preserving the order relation].*

Notice however that the other direction does not have to hold, i.e., it can be that $\mu(t_1) \prec_N \mu(t_2)$ but neither $t_1 \prec_M t_2$ nor $t_2 \prec_M t_1$. Consider the chart in Figure 1, this time interpreted as a template (Figure 5). It specifies that there

are at least 4 events, and that the send event s_1 precedes the receive event r_1, and similarly, s_2 precedes r_2. However, the template does not impose any order between r_1 and r_2. This does not mean that the template would match only charts where r_1 and r_2 are unordered; it merely means that by not imposing such an order it would match charts regardless of any order between these events.

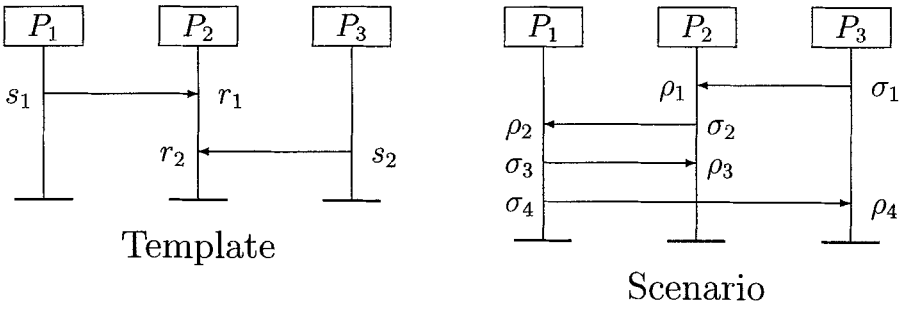

Fig. 5. A template and an MSC

The definition of matching depends on the semantic rules used to translate an MSC into a causal structure. Consider the template M and the MSC N in Figure 5. The corresponding template precedence order, under the above semantics rules, which does not force order between r_1 and r_2, appears in the lower right of Figure 1. The MSC precedence order consists of the chain $\sigma_1 \prec_N \rho_1 \prec_N \sigma_2 \prec_N \rho_2 \prec_N \sigma_3 \prec_N \rho_3$ and the pairs $\rho_2 \prec_N \sigma_3$, $\sigma_3 \prec_N \sigma_4$, $\rho_2 \prec_N \sigma_4$ and $\sigma_4 \prec_N \rho_4$. The embedding function μ of the matching consists of the pairs $\{(s_1, \sigma_3), (r_1, \rho_3), (s_2, \sigma_1), (r_2, \rho_1)\}$.

Consider now a different semantics, which orders receive events on the same process according to their appearance in the MSC. The template precedence order for this case, which is the same as the visual order, appears in lower left of Figure 1. The MSC precedence order now includes also $\rho_1 \prec_N \rho_3$, while the template precedence order includes $r_1 \prec_M r_2$. Under this semantics, the template does not match the MSC. To see this, notice that any embedding function μ must contain at least the pairs $\{(s_1, \sigma_3), (s_2, \sigma_1)\}$ in order to satisfy the process and type matching condition. Because of the message edges, it also has to include the four pairs as under the previous case. But since $r_1 \prec_M r_2$, a match must also satisfy that $\mu(r_1) = \rho_3 \prec_N \rho_1 = \mu(r_2)$, which does not hold.

The following theorem is useful for developing an algorithm for matching templates and charts.

Theorem 1 *If a template M matches an MSC N then for each linearization $\sqsubset_N$ of $\prec_N$ there exists a linearization $\sqsubset_M$ of $\prec_M$ and a homomorphic mapping $\nu : E_M \mapsto E_N$ such that if $e_1 \sqsubset_M e_2$ then $\nu(e_1) \sqsubset_N \nu(e_2)$.*

Proof. Assume that M matches N. Let μ be the embedding mapping. Choose a linearization $\sqsubset_N$ of $\prec_N$, and let $\sqsubset_M = \{(e, f) | \mu(e) \sqsubset_N \mu(f)\}$. We claim that $\sqsubset_M$ is a linearization of $\prec_M$. To see this, assume for the contrary that $e \prec_M f$ but $f \sqsubset_M e$. Then, according to Definition 1, since $e \prec_M f$, it must hold that $\mu(e) \prec_N \mu(f)$. But then, $\mu(e) \sqsubset_N \mu(f)$ and thus, $e \sqsubset_M f$. But since $\sqsubset_M$ is a total order, it cannot hold that both $e \sqsubset_M f$ and $f \sqsubset_M e$. ∎

Thus, it is sufficient to compare a single linearization of the MSC N against the linearizations of the template M. To develop a matching algorithm, we exploit the following standard definitions [2].

Definition 2 *A* slice $S \subseteq E$ *of a causal structure* $\mathcal{O} = \langle E, \prec, L, T, \mathcal{P} \rangle$ *satisfies that for each pair of events* $e_1, e_2 \in E$, *such that* $e_1 \prec e_2$, *if* $e_2 \in S$ *then* $e_1 \in S$.

A slice is often called a *configuration*. The set of slices of a causal structure $\mathcal{O}$ is denoted by $\mathcal{S}(\mathcal{O})$. The pair $\langle \mathcal{S}(\mathcal{O}), \subseteq \rangle$ forms a partial order of slices.

Definition 3 *A* cut *of a causal structure* $\mathcal{O} = \langle E, \prec, L, T, \mathcal{P} \rangle$ *is a maximal set of edges* $C \subseteq \prec$, *satisfying that there exists a slice* $S \subseteq E$ *such that for each edge* $(e_1, e_2) \in C$, $e_1 \in S$ *and* $e_2 \notin S$.

The set of cuts of a causal structure $\mathcal{O}$ is denoted by $\mathcal{C}(\mathcal{O})$. It is easy to see that for each slice $S \in \mathcal{S}(\mathcal{O})$ there is a unique matching cut $C \in \mathcal{C}(\mathcal{O})$. A slice S_2 is an *immediate successor* of a slice S_1 if $S_2 = S_1 \cup \{e\}$ for some event $e \in E$.

To create a systematic search of the linearizations of a template M, one can apply a depth first search as follows: the states of the search are the slices of the template. The search starts with the empty slice. It progresses from a current slice S to its immediate successor slices. When progressing from S to $S \cup \{e\}$, the edge is marked with the event e. It is standard to show that the paths generated in this search correspond to all the linearizations of the partially ordered causality relation $\prec^*$. Figure 6 gives the linearizations of the template in Figure 5.

The graph resulting from the search can be immediately converted into an automaton such that the events labeling the edges of each run form a linearization of the template order. Since a template needs to match only a *subset* of the events of an MSC, each node includes a self loop that allows arbitrary additional events, which are not covered by the template. These edges are marked with the symbol τ. The *template automaton* A_M is a fivetuple $\langle S_M, \longrightarrow_M, \iota_M, F_M, \delta_M \rangle$, where S_M is the set of states, $\longrightarrow_M$ is the transition relation, ι_M is the initial state, F_M is the set of accepting states, and δ_M is the labeling on the edges.

For a chart N, one can construct an automaton $A_N = \langle S_N, \longrightarrow_N, \iota_N, F_N, \delta_N \rangle$, which accepts all the prefixes of one of its linearizations. For example, an automaton for a linearization of the MSC in Figure 5 can be as follows:

$$x_0 \xrightarrow{\sigma_1:P_3} x_1 \xrightarrow{\rho_1:P_2} x_2 \xrightarrow{\sigma_2:P_2} x_3 \xrightarrow{\rho_2:P_1} x_4 \xrightarrow{\sigma_3:P_1} x_5 \xrightarrow{\sigma_4:P_1} x_6 \xrightarrow{\rho_4:P_3} x_7 \xrightarrow{\rho_3:P_2} x_8 \qquad (1)$$

(Notice that there are other linearizations, as, e.g., ρ_3 and σ_4 are not ordered according to the precedence order). For such an automaton, there is one initial state, and all the states are accepting.

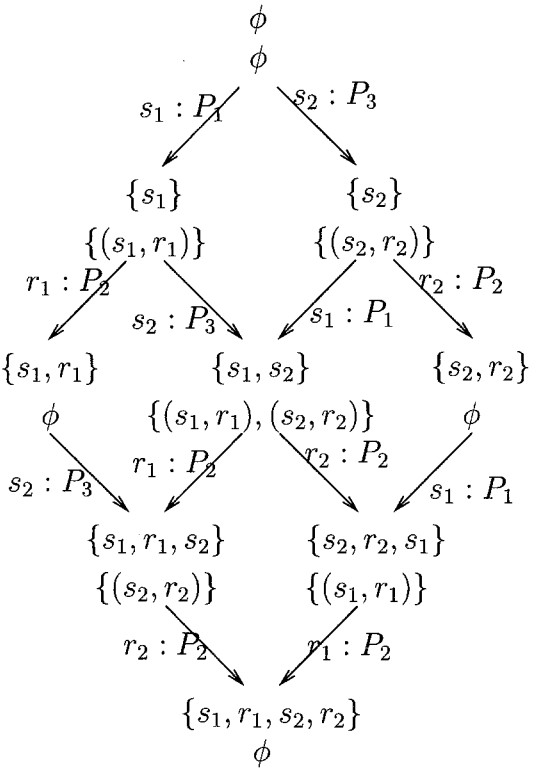

Fig. 6. The cuts/slices graph for the template in Figure 5

The *product automaton* $A_M \times A_N$ consists of the Cartesian product of states $S_M \times S_N$, the transition relation such that $\langle s, t\rangle \longrightarrow_{M\times N} \langle s', t'\rangle$ iff $s \longrightarrow_M s'$ and $t \longrightarrow_N t'$, the initial state $\langle \iota_M, \iota_N\rangle$, the accepting states $F_M \times F_N$, and a labeling function $\delta_{M\times N}$ which labels a transition $\langle s, t\rangle \longrightarrow_{M\times N} \langle s', t'\rangle$ by $\langle \delta_M(s \longrightarrow_M s'), \delta_N(t \longrightarrow_N t')\rangle$.

The *match product* $A_M \bowtie A_N$ defined below is a modification of the product automaton, constructed for the matching algorithm. Each node $\langle s, t, b\rangle$ in the product, contains also a third additional component b besides the pair of states s from A_M and t from A_N, respectively. The component b, called the *bindings*, is a set of triples of the form $(e_1, \rho, e_2) \in E_M \times E_N \times E_M$. Projecting out the middle component from each triple, one obtains the cut that is associated with the template component s. The intuitive meaning is that the template event e_1 is matched with the chart event ρ (while the event e_2 is not matched yet).

Certain rules dictate the transitions of $A_M \bowtie A_N$. Consider such a transition from a state $\langle s, t, b\rangle$ to a state $\langle s', t', b'\rangle$, where $s \longrightarrow_M s'$ and $t \longrightarrow_N t'$:

- The transition is labeled by a pair $\langle \tau, \rho \rangle$, where $\rho \in E_N$. Then, the MSC event ρ does not correspond to any event in the template (the template automaton is doing a self loop).

- The transition is labeled by a pair $\langle e, \rho \rangle \in E_M \times E_N$, where the events e and ρ agree on their type and process. In this case, the following conditions impose the relation between the bindings b and b':

  Adding triples. $(e, \rho, g) \in b' \setminus b$ iff $g \in E_M$ and $e \prec_M g$. [The new triples correspond to new edges (e, g) on the cut corresponding to s', recording that e was matched (with ρ).]

  Forgetting a triple. $(f, \sigma, e) \in b \setminus b'$ iff $\sigma \prec_N \rho$. [Matching e with ρ and matching f with σ preserve the orders, i.e., $f \prec_M e$ and $\sigma \prec_N \rho$.]

It is easy to see from the construction that checking the match between template M and an MSC N can be done by checking the emptiness of the automaton $A_M \bowtie A_N$. The match product accepts at least one sequence iff the template M and the MSC N match. A match between M and N can be obtained from any accepting run with an embedding function μ such that $\mu(e) = \rho$ iff there exists an edge labeled by $\langle e, \rho \rangle$ during the run.

It is simple to extend this to a family of charts embedded in a rational expression or a graph, respectively, as defined in Section 2.2. This relies on the semantic rules for interpreting an MSC to satisfy the following:

> if A and B are two charts, $e \in E_A$ and $f \in E_B$, then it is not the case that $f \prec_{AB} e$.

Under this condition we have:

Lemma 1 *Let A, B be two message sequence charts, with precedence orders $\prec_A \subseteq E_A \times E_A$ and $\prec_B \subseteq E_B \times E_B$, respectively, where $E_A \cap E_B = \phi$. Let $\sqsubset_A$ and $\sqsubset_B$ be linearizations of $\prec_A$ and $\prec_B$, respectively. Then, $\sqsubset_A \cup \sqsubset_B \cup \{(e, f) | e \in E_A \wedge f \in E_B\}$ is a linearization of $\prec_{AB}$.*

Thus, generating an automaton that recognizes at least one linearization for each MSC in a system of MSCs defined using a rational expression or a graph (as defined in Section 2.2) can be obtained by a simple composition of the linearizations of the component MSCs.

3.2 Complexity and Efficiency

The time and space complexity of the algorithm is $O((n/m)^m)$, where m is the size of the template, and n is the size of the checked MSC. Thus, it is exponential in the size of the template, and for a fixed template, polynomial in the size of the MSC. Using a standard binary search argument [12], one can obtain from our description an algorithm that is PSPACE in the size of the template.

We will make now a few comments about choices made, which were affected by the strive for an efficient algorithm.

For efficiency of the matching algorithm, the order $\prec_M$ of a template M should be a reduced order. Thus, if $e_1 \prec_M e_2$ and $e_2 \prec_M e_3$, then e_1 and e_3 should not be ordered by $\prec_M$. To see this, suppose that the send events e_1, e_2 and e_3 are matched against σ_1, σ_2 and σ_3 in an MSC N, respectively. The matching requires that $\sigma_1 \prec_N \sigma_2$, $\sigma_2 \prec_N \sigma_3$. Thus, $\sigma_1 \prec_N \sigma_3$ is implied, without enforcing an order between $e_1 \prec_M e_3$. Thus, there is one less triple, namely that of (e_1, σ_1, e_3) to store and check. Thus, the translation of the visual order of a template into its precedence order is somewhat different than the translation of the visual order of an MSC: in the former case, when adding a pair $e \prec_M f$ to the precedence order, one needs to check that there can be no pairs $e \prec_M g$ and $g \prec_M f$ for some event g.

3.3 Additional constructs

So far, the template provided a subset of the events, to be matched against an MSC (or a graph of MSCs). The order corresponded to precedence order. However, in some cases, one might want to make a stronger assertion about the order. Namely, to express the fact that a pair of events are ordered and no events can appear in between. This case is handled by restricting the self loops on the nodes. Suppose there are two subsequent edges marked with events e and f of the same process, and the template indicates 'immediate order'. Then an edge labeled by τ is not allowed between edges labeled by e and f.

To distinguish between 'immediate' and 'eventual' orders, one can use usual process lines to indicate immediate order, and a broken (dotted, or dashed) process lines to indicate eventual order.

Another extension is to allow annotating events and messages with textual names, and to allow the match of a named event in a template only with an MSC event with the same name.

4 An Implementation

We describe an implementation of the algorithm using the COSPAN [7] model-checking tool. The language S/R (for *selection/resolution*) is the input interface to the COSPAN tool.

The program first translates the template in Figure 5 to the list of pairs of events. Each event indicates by its first letter whether the event is a send or a receive. The message number appears in square brackets (hence a message is a pair of events with the same message number), and the process where this event appears follows a colon. Each line represents a pair of events in the precedence order. The events of the template in Figure 5 are translated into:

```
s[1]:1,r[1]:2
s[2]:3,r[2]:2
```

Similarly, the translation of the MSC in Figure 5 is as follows:

```
s[11]:3,r[11]:2          r[11]:2,s[12]:2
```

```
s[12]:2,r[12]:1        r[12]:1,s[13]:1
s[13]:1,r[13]:2        r[12]:1,s[14]:1
s[14]:1,r[14]:3        s[13]:1,s[14]:1
```

Notice that we added 10 to each event index in order not to confuse between the template and MSC events (the implementation allows to reuse the same numbers for both). Hence, `s[11]` represents the event σ_1.

The program then generates S/R code from these two lists which represent precedence orders of a template and an MSC respectively. This S/R code specifies two parallel processes where the first, called `Tmp`, represents the template automaton and the other, called `Dom`, represents the MSC automaton.

The S/R language allows a specification of a system of parallel processes which move from state to state simultaneously after a non-deterministic selection of current values for *selection* variables. A state is interpreted in COSPAN as a vector of values of *state* variables which are disjoint from the selection variables. A transition from a state to another state is implemented by a set of assignments to the state variables. Each process may have one standard selection variable `#` and one standard state variable `$`. The former is linked to the latter as follows: each value of `$` is explicitly supplied with a permitted range of currently possible values for `#`. The values of variable `$` may often be thought of as 'state positions'.

The states of the template and MSC automata described in Section 3 (see Figures 6 and Formula 1), are mapped one-to-one into state positions of S/R processes `Tmp` and `Dom`, respectively. The coordination of these two S/R processes models the match of a template automaton against MSC as explained next: In each of the two processes, at each of the state positions, the permitted range of the selection variable `#` is the set of the next send/receive events that generate a successor for the current slice. The process `Dom` implements an automaton that recognizes a linearization of the MSC, and therefore deterministically progresses from one state position to another, keeping the executed event as a value of variable `Dom.#`. Below is the transition structure of `Dom` process for the MSC automaton:

```
trans
NoEvent{s11} -> s11: true;
    s11{r11} -> r11: true;
    ...
    s14{r14} -> r14: true;
    r14{r13} -> r13: true;
    r13{NoEvent} -> $: true;
```

Note that the value of selection variable `Dom.#` placed in braces follows the current state position. The state position is named after the most recent event encountered. For example, the state `s11` indicates that the last event was `s11` (which represents σ_1).

The process `Tmp` whose state positions correspond to the template automaton slices may either self-loop at a current state position or non-deterministically progress to the next state position N. The latter case is accomplished iff the selected event `E` fits the matching condition described in section 3. This guarded transition is expressed in S/R as follows:

```
->N: (#=E)*MatchCond_E
```

where `MatchCond_E` is an S/R predicate that expresses the matching condition and the symbol $*$ stands for logical *and* ($\wedge$). If there is no such event `E`, the process `Tmp` self-loops, thus waiting for the process `Dom` to execute an appropriate event. As an example, consider the transitions corresponding to the middle slice $\{s_1, s_2\}$, with the cut $\{(s_1, r_1), (s_2, r_2)\}$, appearing in Figure 6:

```
4{Tau,r1,r2}
  -> 2:(#=r1)*MatchCond_r1
  -> 1:(#=r2)*MatchCond_r2
  -> $: else;
```

Process `Tmp` allows three selections out of this state position (named `4`): one is an attempt for matching the event r_1 (translated into `r[1]:2` and then into `r1`), another for matching r_2 (`r[2]:4`, then `r2`), and the third is a τ move, hence remaining in the same state, i.e. looping back to the state position `$`. The self-looping also executes if a selection for matching an event (`r1` or `r2`) does not fit the corresponding matching predicate (`MatchCond_r1` or `MatchCond_r2`, respectively). The matching predicates are defined as S/R macros. For example,

```
macro MatchCond_r1 :=
 (Dom.NxtStProc=2)*(Dom.NxtStType=r)*
        (Dom.CntrpProc=1)*(Img_s1_predsNxtDomSt)
```

This checks that the MSC event currently executed agrees with the selected template event `r1` on the process `(Dom.NxtStProc=2)` (which is P_2 for both), and on the type `(Dom.NxtStType=r)`. Furthermore, the process of the corresponding send event in the MSC matches the process of the corresponding send event in the template `(Dom.CntrpProc=1)` (process P_1). In addition, all the predecessors of the selected template event must have matched with the MSC events, which are related by the MSC precedence order with the MSC event currently executed. This is checked by the predicate `Img_s1_predsNxtDomSt`. The latter is defined via straightforward application of the MSC precedence order to variables `Img_s1`, described below, and `Dom.#`.

The same matching conditions that allow the process `Tmp` to move from the above slice by matching the event `r1`, are used to bind the currently executed MSC event, which is accessible as value of `Dom.#`, to the state variable `Img_r1`. This is done using the first line in the following assignment which is a part of `Tmp` process:

```
asgn Img_r1 -> Dom.# ? (#=r1)*MatchCond_r1 |
               NoDomEvent ? ~(Event_r1_inCut) | Img_r1
```

The syntax of this assignment statement is as follows: the variable to be assigned appears before the arrow. Then we have pairs of *value ? guard*, separated by the alternative (|) symbol. Such an assignment is global, thus it is tested and executed in every transition. The second alternative of the assignment corresponds to 'forgetting' a match (by storing the special value `NoDomEvent`). The symbol

$\sim$ is the negation symbol. The condition `Event_r1_inCut` is true exactly in the cases where `r1` is in the cut (it is identically *false* in our example).

In COSPAN, automata are defined over *infinite* sequences. COSPAN detects an accepting sequence by searching for cycles that satisfy its acceptance conditions in the state space generated for the coordinating processes. Such a cycle is reported as a "bad cycle". For model-checking, it means the existence of a counter example for the checked property. The checked property which is embedded into the generated S/R code is "the process `Tmp` never reaches its final state position (i.e. the final slice)", and the implementation forces an artificial self-loop at this state position. So, the corresponding "bad cycle" means a match. The matching pairs appear in the COSPAN report for such a bad cycle as the values of the variables `Dom.#` and `Tmp.#`, when `Tmp.#` is not `Tau` (corresponding to a τ move). The program extracts these values and outputs a matching table. For the above example, we obtain:

```
s[2] -> s[11]    s[1] -> s[13]    r[1] -> r[13]    r[2] -> r[11]
```

Acknowledgement

The authors would like to thank Bob Kurshan and Mihalis Yannakakis for many illuminating discussions on the subject.

References

1. R. Alur, G.J. Holzmann, D. Peled, An Analyzer for Message Sequence Charts, *Software Concepts and Tools*, Vol. 17, No. 2, 1996, pp 70-77.
2. E. Best, R. Devillers, Sequential and concurrent behaviour in Petri Net theory, *Theoretical Computer Science* 55 (1987), 87–136.
3. Tz. Elrad, N. Francez, Decomposition of distributed programs into communication closed layers, *Science of Computer Programming* 2 (1982), 155–173.
4. G.J. Holzmann Early Fault Detection Tools, *Software Concepts and Tools*, Vol. 17, No. 2, 1996, 63-69.
5. ITU-T Recommendation Z.120, Message Sequence Chart (MSC), March 1993.
6. W. Janssen, J. Zwiers, Protocol design by layered decomposition, a compositional approach, *Proceedings of formal techniques in real-time and fault-tolerance systems 1992*, LNCS 571, Springer, 307-326.
7. R.P. Kurshan, *Computer-Aided Verification*, Princeton University Press, 1994.
8. S.C. Kleene, Representation of events in nerve nets and finite automata, *Automata Studies, annals of math studies* 34, Princeton University Press, 1956.
9. A. Mazurkiewicz, Trace theory, *Advanced course on Petri nets*, Bad Honnef, Germany, 1987, LNCS 254, 269-324.
10. M. Nielsen, G. Plotkin, G. Winskel, Petri Nets, Event Structures and Domains, Part I, Theoretical Computer Science 13(1981), 85–108.
11. V. Pratt, Modeling concurrency with partial orders, *International Journal of Parallel Programming* 15 (1986), 33–71.
12. W. J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. on Computer and System Sciences*, 4 (1970), 177-192.