An Analyzer for Message Sequence Charts

Rajeev Alur*
Gerard J. Holzmann*
Doron Peled*

ABSTRACT Message sequence charts (MSCs) are used in the design phase of a distributed system to record intended system behaviors. They serve as informal documentation of design requirements that are referred to throughout the design process and even in the final system integration and acceptance testing. We show that message sequence charts are open to a variety of semantic interpretations. The meaning of an MSC can depend on, for instance, whether one allows or denies the possibility of message loss or message overtaking, and on the particulars of the message queuing policy to be adopted.
We describe an analysis tool that can perform automatic checks on message sequence charts and can alert the user to the existence of subtle design errors, for any predefined or user-specified semantic interpretation of the chart. The tool can also be used to specify time constraints on message delays, and can then return useful additional timing information, such as the minimum and the maximum possible delays between pairs of events.

Proc. 2nd Intern. Workshop. on Tools and Algorithms for the construction and Analysis of Systems, TACAS96, Passau, Germany, March 1996, Lecture Notes in Computer Science, Vol. 1055, pp. 35–48.

1 Introduction

Message sequence charts (MSCs)—also known as time sequence diagrams, message flow diagrams, or object interaction diagrams—are a popular visual formalism for documenting design requirements for concurrent systems. MSCs are often used in the first attempts to formalize design requirements for a new system and the protocols it supports. MSCs represent typical execution scenarios, providing examples of either normal or exceptional executions of the proposed system.

Like any other aspect of the design process, MSCs are amenable to errors,

*AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974.
Email: {alur,gerard,doron}@research.att.com

the most common of which are *race conditions*. A race condition exists when two events appear in one (visual) order in the MSC, but can be shown to occur in the opposite order during an actual system execution. These conflicts can result from incorrect or incomplete assumptions about chains of dependencies in the design, or from conflicting semantic assumptions about the underlying communication system. The ambiguities may lead to unspecified reception errors, deadlocks, loss of messages, and other types of incorrect behavior in the final system. Some semantic interpretations of the MSC may permit the occurrence of race conditions, while others may circumvent them. The specific version of the semantics used is influenced by the underlying communication architecture that will be chosen for the final design. The semantics are different, for instance, when processes have a single input queue or multiple queues, and it can depend on whether or not the messages are stored in FIFO order.

We describe some generic algorithms for analyzing message sequence charts, and a tool that implements them. The tool allows the user to construct and edit message sequence charts interactively, in graphical form, and to store these charts in either Z.120 textual form [5], or in graphical form as PostScript files. The tool provides the user with a menu of possible semantic interpretations of a given MSC, and can detect conflicts such as causality cycles and race conditions.

When the user specifies additional information, the tool can also perform timing analysis. The additional information consists of user-defined bounds on message delays, and bounds on delays between successive send operations. The analyzer can check whether the timing constraints are consistent, and can derive additional information such as the minimum and the maximum expiration times for timers.

The analyzer can serve as a convenient means to integrate formal verification techniques into the design process, in a way that is almost invisible to the users. The MSC analyzer, for instance, can be extended to produce formal models in the input language of standard model checkers, such as SPIN [4], to permit more detailed analyses of a design.

There have been several attempts to define an appropriate formal semantics for MSCs, e.g., [6], [7]. These approaches provide semantics definitions that correspond to, what we will define to be, the *visual order* of events. Our approach allows the user to formalize more specifically the assumptions that the user can make about the underlying (or target) architecture of the system, and compare the resulting semantics against the visual order.

2 Message Sequence Charts and their Semantics

A sample MSC is shown in Figure 1. For illustrative purposes, it reflects only a small number of the possible features. For a more complete descrip-

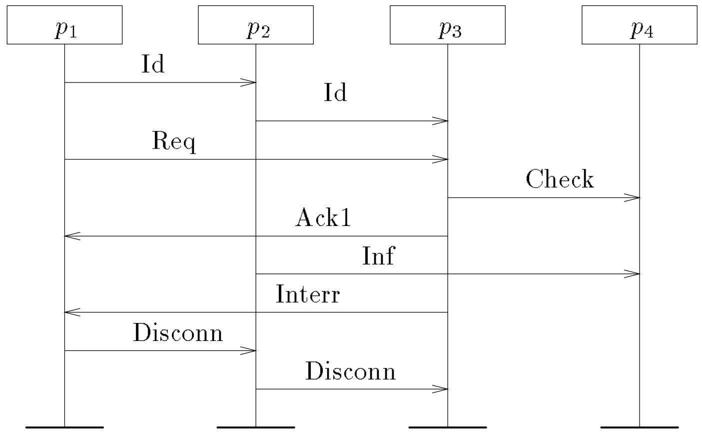


FIGURE 1. A message sequence chart

tion of MSCs, refer to the ITU recommendation Z.120 [5]. The tool we will describe supports all the features of basic message sequence charts. As yet, it does not include additional features such as creation or destruction of processes, co-regions (to be discussed below), and sub-MSCs.

Vertical lines in the chart correspond to asynchronous processes or autonomous agents. Messages exchanged between these processes are represented by arrows. The tail of each arrow corresponds to the event of sending a message, while the head corresponds to its receipt. Arrows can be drawn either horizontally or sloping downwards, but not upwards.

2.1 Formalization

To formalize MSCs and allow their analysis, consider the MSC of Figure 2. It contains 3 processes, numbered from left to right p_1, p_2, p_3. For each process p in the system there is a vertical line which defines a local visual order, denoted $<_p$, on all the events belonging to p. Each event is either a send or a receive event, and belongs to one specific process. The events of sending and receiving messages are labeled by s_1, s_2, s_3, r_1, r_2, and r_3. For each send event, there exists a matching receive event, and vice versa. This means that, in the charts that we will use here, there are no anonymous *environment* processes. If an environment process is used, it is represented by a vertical line in the MSC. As we will see in the sequel, the actual order of occurrence of any two events in the MSC may or may not correspond to the visual order in the chart, depending on the semantic interpretation that is used.

A message sequence chart M defines a labeled directed acyclic graph with the following components:

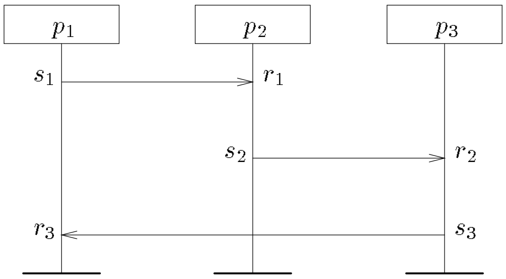


FIGURE 2. A simple MSC

- *Processes:* A finite set P of processes.
- *Events:* A finite set S of send events and a finite set R of receive events such that $S \cap R$ is empty. The set $S \cup R$ is denoted by E.
- *Process Labels:* A labeling function $L : E \mapsto P$ that maps each event e to a process $L(e) \in P$. The set of events belonging to a process p is denoted by E_p.
- *Send-receive Edges:* A *compatibility* bijection $c : S \mapsto R$ such that each send event s is mapped to a unique receive event $c(s)$ and each receive event r is mapped to a unique send event $c^{-1}(r)$.
- *Visual Order:* For every process p there is a local total order $<_p$ over the events E_p which corresponds to the order in which the events are displayed. The relation

  $$< \triangleq (\cup_p <_p) \cup \{ (s, c(s)) \mid s \in S \}$$

  contains the local total orders and all the edges, and is called the *visual order*.

The visual order defines an acyclic graph over the events since send-receive edges cannot go upwards in the chart. The visual order does not necessarily reflect the semantics of the MSC. Although some event e may appear before an event f in the visual order, this may be only due to the two dimensionality of the diagram; it may be that e and f can in practice occur in either order. An automated scenario analyzer can, then, warn the designer that events may occur in an order that differs from the visual one.

2.2 Ambiguities

To illustrate the potential ambiguities of MSC specifications, two questions need to be addressed in assigning semantics to MSCs:

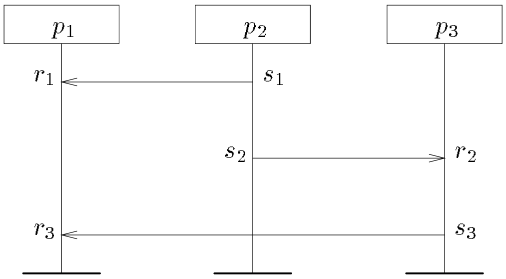


FIGURE 3. Another simple MSC

1. Which causal precedences are enforced by the underlying architecture?

2. Which causal precedences are likely to be inferred by the user?

Any discrepancy between the answers to the above two questions could lead to design errors and requires the user's attention.

Consider Figures 2 and 3. In Figure 2, it is reasonable to infer that receive event r_3 occurs after 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

$$s_1 \ll r_1 \ll s_2 \ll r_2 \ll s_3 \ll r_3$$

where the symbol $\ll$ represents causal precedence.

However, it is not clear if the receive event r_1 precedes the receive event r_3 in Figure 3. 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 persuaded 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, if it cannot distinguish between the two messages and merely assumes that one will always precede the other, it may end up deriving information from the wrong message.

The ITU Z.120 recommendation contains a mechanism for defining that the order of occurrence of events is either unknown or immaterial, using *co-regions*. For the user, however, it can be hard to assess correctly where precisely co-regions are required, where they are redundant, or even invalid. The analysis tool can identify the regions accurately in all cases.

The semantics of the enforced order can also depend on the underlying architecture of the system. Consider, for instance, two subsequent messages, sent one after the other from one process to the other. The arrival of the messages in the same order in which they were sent is guaranteed only if the architecture guarantees a FIFO queuing discipline. When this is not guaranteed, an alternative semantics in which messages can overtake each other is called for.

2.3 Interpreted MSCs

As discussed above, the correct semantic interpretation may depend on many things that cannot be standardized, such as the particulars of the underlying architecture or the communication medium and queueing disciplines that are used. We therefore adopt a user-definable semantics, and predefine only a small number of reasonable semantic interpretations.

There are three types of causal precedences that we will distinguish in this paper:

The *visual* order $<$. As explained in Section 2.1, the visual order corresponds to the scenario as drawn.

The *enforced* order $\ll$. This order contains 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 enforced 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.

The *inferred* order $\sqsubset$. Events that are ordered according to the inferred order are likely to be assumed by the user to occur in that order. A tool can check that the inferred order is valid by computing the transitive closure of the enforced order.

The enforced and the inferred orders can both be defined as subsets of the visual order, i.e., $(\ll \cup \sqsubset) \subseteq <$. Different semantic interpretations correspond to different rules for extracting the enforced and inferred order from the visual order. For example, a pair $(s, c(s))$ of a send and a corresponding receive event is always in the enforced order. On the other hand, a pair (r_1, r_2) of receive events in the visual order may appear in either the enforced order or in the inferred order, but it need not appear in either.

Formally, an *interpreted message sequence chart* M consists of the following components:

- An MSC $\langle P, S, R, L, c, \{<_p | p \in P\}\rangle$,
- For every process p, a binary relation $\ll_p$ over E_p: $e \ll_p f$ means that event e *is known* to precede event f. It is required that $\ll_p$ is a subset of the visual order $<_p$. The enforced order $\ll$ is

$$(\cup_p \ll_p) \cup \{(s, c(s)) | s \in S\}.$$

- For every process p, a binary relation $\sqsubset_p$ over E_p: $e \sqsubset_p f$ means that event e *is assumed* to precede event f. It is required that $\sqsubset_p$ is a subset of the visual order $<_p$. The inferred order $\sqsubset$ is $\cup_p \sqsubset_p$.

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 transitive closure. It can then be checked whether $\sqsubset$ is a subset of $\ll^*$. If this is not the case, there is a conflict between the enforced and the inferred orders, and the user is likely to make an invalid inference about the behavior of the system. For example, the race conflict in Figure 3 corresponds to the interpretation that $\ll$ is $\{(s_1, r_1), (r_1, s_2), (s_2, r_2), (r_2, s_3), (s_3, r_3)\}$, while (s_1, r_3) is in $\sqsubset$.

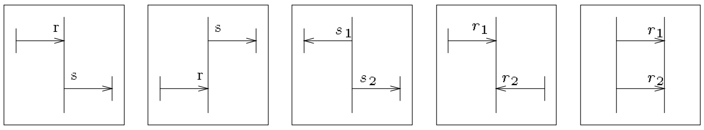
Observe that since the visual order is acyclic, so is the relation $\ll^*$ due to the requirement that each $\ll_p$ is a subset of $<_p$. Also note that the two orders $\ll$ and $\sqsubset$ cannot conflict since both are consistent with the visual order.

There is more than one reasonable semantic interpretation of an MSC. We consider four sample choices, each tied to a different choice for the underlying architecture. Consider two events of the same process p. Each event is either a send or a receive event, with a matching receive or send event in some other process. Figure 4 illustrates the corresponding five cases that are relevant to our default set of interpretations.

Four default choices for the relations $\ll$ and $\sqsubset$ are indicated, as enumerated below. Cases A, B, and C, share the same interpretations in all four defaults. Cases A and C formalize the notion that a send event is a controlled event, that is only issued when the preceding events in the visual order have occurred. The order is therefore enforced in both cases, under all semantic interpretations. In case B, the inference is made that the receive event r can happen only after the send event s to account for the case where s is meant to provoke the reception r. Cases D and E distinguish between the the case when the two matching send events for two receive events that arrive to the same process p belong to the same process q or to two different processes q and r, and are interpreted differently in different defaults:

1. *Single FIFO-queue per process:* Each process p has a single FIFO queue to store all the messages received by p. Messages received by p from the same source arrive in the order in which they are sent (case E), but messages received by p from different sources (case D) need not arrive in the order sent. The inferred order of receive events corresponds to the visual order. In this semantics, if a process is waiting to receive a message r_1, and if r_2 arrives before r_1, then r_2 may be lost, or a deadlock may occur.

2. *One FIFO queue per source:* Each process p has one FIFO queue for every process q to store all the messages received by p from q.



	A	B	C	D	E
Single Fifo	$r \ll s$	$s \sqsubset r$	$s_1 \ll s_2$	$r_1 \sqsubset r_2$	$r_1 \ll r_2$
Fifo/source	$r \ll s$	$s \sqsubset r$	$s_1 \ll s_2$	none	$r_1 \ll r_2$
NonFifo	$r \ll s$	$s \sqsubset r$	$s_1 \ll s_2$	$r_1 \sqsubset r_2$	$r_1 \sqsubset r_2$
NonFifo/source	$r \ll s$	$s \sqsubset r$	$s_1 \ll s_2$	none	$r_1 \sqsubset r_2$

FIGURE 4. Defaults for interpreted MSCs

Since messages received from different sources are stored in different buffers, no order is inferred for the two receives in case D. This is because with multiple queues, a process has direct access to the first message arriving from each process, and the relative order of two messages arriving from difference processes is unimportant. If the wrong message arrives first, the receiving process would still be able to wait for the arrival of the other message, and after processing the second one, the first one would still be in its own message input queue.

3. *Single Non-FIFO queue per process:* The order in which messages are received is not necessarily the same as the order in which the messages are sent. Thus, for case E, no order between r_1 and r_2 is known. The inferred order between receive events corresponds to the visual order.

4. *One Non-FIFO per source:* Each process p has one FIFO queue for every process q to store all the messages received by p from q. Due to non-FIFO nature, for case E, the order among receives is only inferred, and not necessarily enforced. Due to multiple queues, for case D no order is inferred for receives from different sources.

Alternative interpretations may be provided for different choices of the underlying queuing model. The user can also be given an explicit override capability, to make different semantic choices for specific, user-selected, event pairs.

3 The Analysis of MSCs

Consider an interpreted MSC with visual order $<$, enforced order $\ll$, and inferred order $\sqsubset$. To find inconsistencies the transitive closure $\ll^*$ of the enforced order is computed and compared against the inferred order.

> *Race Condition:* Events e and f from the same process p are said to be in a race if $(e \sqsubset f)$ but (not $e \ll^* f$).

The MSC analysis problem is to compute all the races of a given interpreted MSC.

The causality relations $\ll$ and $\ll^*$ define partial orders over the set E of all events in M. Once the transitive closure is computed, conflicts can be identified by examining each event pair in the inferred order.

Due to the special structure of our problem, we can use the following algorithm to compute the transitive closure, at a lower cost than the standard Floyd-Warshall algorithm.

Assume the MSC has n events. Since there are no cycles, we can number the events $1 \ldots n$, such that the numbering defines a total order that is consistent with visual order $<$. The numbering can be done in time $\mathcal{O}(n)$, using a standard topological sort algorithm (see e.g., [10]). A boolean two-dimensional matrix C is used to store the pairs in $\ll^*$. All entries of C are initially false.

Algorithm 1:
for $e := 1$ **to** n **do**
  for $f := e - 1$ **downto** 1 **do**
    if *not* $C[f][e]$ and $f \ll e$ **then**
      $C[f][e] := true$;
      for $g := 1$ **to** $f - 1$ **do**
        if $C[g][f]$ **then** $C[g][e] := true$

In this algorithm, the value of each of the n^2 entries in C can change from false to true at most once. Call event f an *immediate predecessor* of event e if $f \ll e$ and there is no event g such that $f \ll g \ll e$. Observe that the innermost loop of the algorithm is executed for a pair (e, f) only if the event f is an immediate predecessor of the event e.

Theorem 3.1 *Given an interpreted MSC with n events. If relation $\ll$ contains ℓ pairs (f, e) such that event f is an immediate predecessor of event e, then the computational complexity of Algorithm 1 is $n^2 + \ell n$.*

For the default choices of Figure 4, ℓ is bounded by $2n$, which means that for these choices the computational complexity of Algorithm 1 is $O(n^2)$.

4 MSCs with Timing Constraints

In this section, we describe an extension of MSCs to specify timing constraints on a message flow. As an example, consider the MSC in Figure 5. The label $[1, 2]$ on the edge from s_1 to r_1 specifies the lower and upper bounds on the delay of message delivery. The label $[5, 6]$ on the vertical line from r_1 to s_2 specifies bounds on the delay between r_1 to s_2, and models an assumption about the speed of process p_2. The event *set_timer*

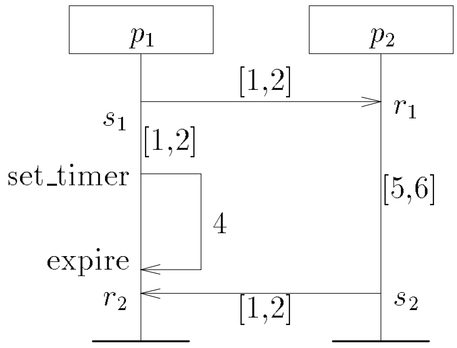


FIGURE 5. An MSC with timing constraints

corresponds to setting a timer which expires after 4 time units. The timing information, in this case, is consistent with the visual order of the two receive events *expire* and r_2. In fact, we can deduce that the timer will always expire before the receive event r_2. Thus, the timing information can be used to deduce additional causal information, or to rule out possible race conflicts. It can also be used to compute maximum and minimum delays between pairs of events. For instance, the separation between the events *expire* and r_2 is at least 1 and at most 5.

Let R^+ be the set of nonnegative real numbers, and let us consider intervals of R^+ with integer endpoints. Intervals may be open, closed or half-closed, and may extend to infinity on the right. Examples of intervals are $(0, \infty)$, $[2, 5]$, $(3, 7]$, where the round brace indicates an open interval, and the square brace a closed one. The set of intervals is denoted by I.

A *timed MSC M* consists of

- An interpreted MSC with enforced order $\ll$ and inferred order $\sqsubset$.
- A timing function $T_\ll : \ll \mapsto I$ that maps each pair (e, f) in the enforced order $\ll$ to an interval $T_\ll(e, f)$. This function models the known timing relationships: the event f is known to occur within the interval $T_\ll(e, f)$ after the event e.
- A timing function $T_\sqsubset : \sqsubset \mapsto I$ that maps each pair (e, f) in the inferred order $\sqsubset$ to an interval $T_\sqsubset(e, f)$. This function models the timing constraints that the user wants to check for consistency.

A *timing assignment* for a timed MSC M is a function $T : E \mapsto R^+$ that assigns, to each event e, a time-stamp $T(e)$ such that for every pair (e, f) in the enforced relation $\ll$ the time difference $T(f) - T(e)$ belongs to the interval $T_\ll(e, f)$. Thus, a timing assignment gives the possible times at which events may occur. A sample timing assignment for the MSC of

Figure 5 is

$$\begin{array}{lll} T(s_1) = 0, & T(set\_timer) = 1.5, & T(r_1) = 2 \\ T(expire) = 5.5, & T(s_2) = 7, & T(r_2) = 8. \end{array}$$

As before, the user may choose the defaults for the relations $\ll$ and $\sqsubset$. The default timing function $T_{\ll}$ maps each pair (e, f) in $\ll$ to the interval $(0, \infty)$.

Timed MSCs can also contain three types of design problems:

1. *Timing Inconsistency:* There exists no timing assignment for the MSC.
2. *Visual Conflicts:* A pair (e, f) of events belonging to the same process p is said to be a visual conflict of the timed MSC if f appears before e in the visual order ($f <_p e$) but in every timing assignment T, e happens before f according to T.
3. *Timing Conflicts:* A pair (e, f) of events is said to be a timing conflict of the timed MSC if e is assumed to occur before f ($e \sqsubset f$), but there is a timing assignment T such that the time difference $T(f) - T(e)$ does not belong to the interval $T_{\sqsubset}(e, f)$.

Timing inconsistency corresponds to an unsatisfiable set of timing constraints. The visual conflict corresponds to the case when the timing constraints imply that the event e always precedes f, in an order opposite to their visual order. Timing conflict corresponds to the case that the inferred bounds are not necessarily satisfied by the timing assignments. The MSC of Figure 5 has no conflicts. Observe that timing imposes additional ordering, and hence, it may be the case that the underlying interpreted MSC has races, but the timed MSC has no conflicts.

The analysis problem for timed MSCs is defined as follows. The input to the timed MSC analysis problem consists of a timed MSC M. If M has timing inconsistency then the output reports inconsistent specification. If M is consistent then the answer to the MSC analysis problem is the set of all visual and timing conflicts.

The timing constraints imposed by the timing function $T_{\ll}$ are linear constraints, where each constraint puts a bound on the difference of two variables. Solving such constraints can be reduced to computing negative-cost cycles and shortest distances in weighted digraphs [9].

The analysis can include both strict and nonstrict inequalities. In order to deal with different types of bounds uniformly, the cost domain D can be defined to be $Z \times \{0, 1\}$, where Z is the set of all integers (such analysis is typical of algorithms for timing verification, see, for instance, [1, 2]). The costs of the edges of the graph is from the domain D. To compute shortest paths, we need to add costs and compare costs. The ordering $\prec$ over D is the lexicographic ordering: $\langle a, b\rangle \prec \langle a', b'\rangle$ iff (1) $a < a'$, or (2) $a = a'$ and

$b < b'$. The addition is defined by $\langle a,b\rangle + \langle a',b'\rangle = \langle a+a', b+b'\rangle$ (note that $+$ over the boolean component is disjunction). A strict inequality $x-y<a$ is now written as $x-y \leq \langle a,1\rangle$ and a nonstrict inequality $x-y\leq a$ is now written as $x-y\leq \langle a,0\rangle$

Given a timed MSC M, define a weighted digraph G_M as follows. The set of vertices of G_M is the set E of events. The cost of the edge from an event e to an event f gives an upper bound on the difference $T(e)-T(f)$ for a timing assignment for M. Consider a pair (e,f) in the enforced order. If $T_{\ll}(e,f)=[a,b]$, the graph G_M has an edge from e to f with cost $\langle -a,0\rangle$, and from f to e with cost $\langle b,0\rangle$. If $T_{\ll}(e,f)=(a,b]$, the graph G_M has an edge from e to f with cost $\langle -a,1\rangle$, and from f to e with cost $\langle b,0\rangle$. If $T_{\ll}(e,f)=[a,\infty)$ then the graph G_M has an edge from e to f with cost $\langle -a,0\rangle$, and there no edge from f to e. The cases $[a,b)$, (a,b), and (a,∞) are handled similarly.

Lemma 4.1 *The timed MSC M is timing inconsistent iff the graph G_M has a negative cost cycle.*

Suppose M is timing consistent. Let d_{ef} be the length of the shortest path from e to f in the graph G_M (let d_{ef} be ∞ if no such path exists). The paths in G_M, then, represent all the timing assignments for M:

Lemma 4.2 *Let M be a consistent timed MSC. A function $T: E \mapsto R^+$ is a timing assignment for M iff $T(e)-T(f) \prec d_{ef}$ for all events e,f.*

Consequently, a pair (e,f) of events belonging to a process p with $e <_p f$ is a visual conflict iff there is a path from f to e with negative cost (i.e. $d_{fe} < 0$). Let (e,f) be a pair of events in $\sqsubset$. The pair (e,f) is a timing conflict iff the interval $T_{\sqsubset}(e,f)$ is included in the interval $[-d_{ef}, d_{fe}]$. It is clear that the timed MSC analysis problem can be solved by computing the shortest paths in G_M. To compute shortest paths, we use the classical dynamic programming algorithm [3, 11]. This immediately leads to the following theorem:

Theorem 4.3 *Given a timed MSC M with n events the timed MSC analysis problem is solvable in time $O(n^3)$.*

5 An MSC Analysis Tool

In this section, we briefly describe the features of the message sequence chart analyzer that we have implemented to illustrate these ideas. The graphical interface to the MSC analyzer was written in Tcl/Tk [8]. The analyzer itself was written in ANSI standard C.

The most important features of the tool can be summarized as follows.

- The tool allows the user to construct, edit, and analyze MSCs interactively. The charts may be stored in the ITU standard form (Z.120), in textual form as conventional annotated scenarios, or in graphical form, as PostScript files. Annotations to the MSC can be entered in *comment* boxes that become part of the scenario as displayed.
- For the online analysis of interpreted MSCs, the tool supports the four pre-defined semantics choices listed in Figure 4 through menu choices. Other user-defined semantics can easily be incorporated.
- The analysis for race conditions is invoked by clicking on a button labeled 'Check..'. A menu is then created listing all conflicts that can occur for the chosen semantic interpretation of the chart. By selecting a conflict from a menu-list, the corresponding event pair is highlighted in the chart. The user can also set preferences so that only certain types of conflicts (eg. between two receives, or between a send and a receive) are entered into the conflict menus.
- The user can also select an event e, with a mouse click, and ask the tool to identify all related (or optionally all unrelated) events. Related are all those events that necessarily precede or follow e in the partial order $\ll^*$. The two types of events (i.e., following or preceding the selected event) are marked in different colors.
- For timing analysis, the user can annotate the chart with intervals, both on message transmissions and on local process states (see Figure 5). Timing conflicts, for the chosen semantic interpretation, are requested as before, with a mouse click.
- The user can also select an event e, again with a mouse click, and ask the tool to identify for every related event f the interval in which f may happen relative to e. This capability can be used, for instance, to identify the required upper and lower bounds for timer expirations.

The runtime requirements to perform an exhaustive analysis of a scenario are negligible for even large MSCs (in the order of 10^3 events, spanning ten to twenty pages when printed). The analysis tool therefore runs comfortably on even small laptop computers. The tool has been applied successfully to detect race conditions in several routine industrial MSC applications.

The tool can be used to analyze cyclic scenarios by unfolding the MSCs a finite number of times before the analysis begins. If there is a simple cycle, i.e., the complete scenario can repeat, then it is sufficient to analyze only two subsequent copies of the MSC. Thus, in this case there is no need for special machinery: the user can check for race conditions by importing the same MSC twice, one after the other. This will create two subsequent copies, with events of the second copy in process p ordered to appear later than events of the first copy in p in the new local order $<_p$.

6 Conclusions

We have shown that message sequence charts are sensitive to various semantic interpretations. Under different semantics, different race conditions may occur.

We have proposed and implemented a tool which can be used to analyze message sequence charts to locate and visualize design errors as early as possible in a design cycle. The tool conforms to ITU recommendation Z.120. We have noted that extensions of the tool, to gently integrate formal verification techniques further into the design process, are possible. It is our intention to use the formal representation of MSCs described here as a vehicle for exploring such extensions.

Acknowledgements: We thank Chuck Kalmanek, Bob Kurshan, and Mihalis Yannakakis for many fruitful discussions. We are also grateful to Brian Kernighan, who developed a port of the MSC analyzer for Windows PCs.

7 References

[1] R. Alur, A. Itai, R.P. Kurshan, M. Yannakakis. Timing verification by successive approximation. *Information and Computation* **118(1)**, pp. 142–157, 1995.

[2] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, LNCS 407, pp. 197–212, 1989.

[3] R.W. Floyd. Algorithm 97 (Shortest Path). *Communications of the ACM* **5** (1962), pp. 365.

[4] G.J. Holzmann. *Design and Validation of Computer Protocols*, Prentice Hall Software Series, 1991.

[5] ITU-T Recommendation Z.120, Message Sequence Chart (MSC), March 1993. (Includes [7] as Annex B.)

[6] P.B. Ladkin, S. Leue. What do message sequence charts mean. In *Formal Description Techniques*, VI 1995 (FORTE'95), Elsevier, pp. 301–315.

[7] S. Mauw, M.A. Reniers. An algebraic semantics of basic message sequence charts. *The Computer Journal*, **37(4)** (1994).

[8] J. Ousterhout. *Tcl and the Tk toolkit*, Addison-Wesley, 1994.

[9] C.H. Papadimitriou, K. Steiglitz. *Combinatorial Optimization–Algorithms and Complexity*, Prentice-Hall, 1982.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*, MIT press, 1990.

[11] S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, **9** (1962), pp. 11-12.