# A Toolset for Message Sequence Charts

Doron A. Peled

Bell Laboratories
700 Mountain Ave.
Murray Hill, NJ 07974, USA
email: doron@research.bell-labs.com

**Abstract.** Message Sequence Charts (MSCs) are a popular graphical notation for describing communication protocols. MSCs enjoy an international standard (ITU-Z120) and a growing number of tools include an MSC interface for either displaying simulation or verification results, or testing the inclusion of a particular scenario in the design. We describe here a toolset that was developed to help designing a system using the MSC notation. The toolset allows creating an MSC description of a design for communication systems, and performing some verification tasks on the design.

The MSC notation is becoming a popular notation for describing communication protocols. It enjoys an international standard called ITU-Z120 [4]. MSCs are among the standard description techniques used for designing communication systems [2], and a growing number of tools include an MSC interface [8].

Each MSC describes a scenario where some processes communicate with each other. Such a scenario includes a description of the messages sent and received and the ordering between them. 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 (see Figure 1). Since a communication system usually includes many such protocols, a high level description allows combining MSC scenarios together. This description consists of a graph, where each node contains one MSC. Each path in this graph, starting from a designated initial state, corresponds to a single scenario.

In this paper, we describe a toolset that allows performing the early design of a communication system using the notation of message sequence charts. These tools allow applying simple verification tasks to the MSC design. Since the early design is quite abstract, often avoiding detailed information such as the value of particular process variables, the need for further verification at later stages is not eliminated. However, finding design errors early in the development process is very cost-effective and is rather efficient.

The first tool, called Msc [1], supports representing message sequence charts. It allows both a graphical description of the MSCs, and its standard textual representation [4]. Thus, an MSC can be obtained by either drawing it using the graphical interface, or by typing it in its standard syntax. This approach has the

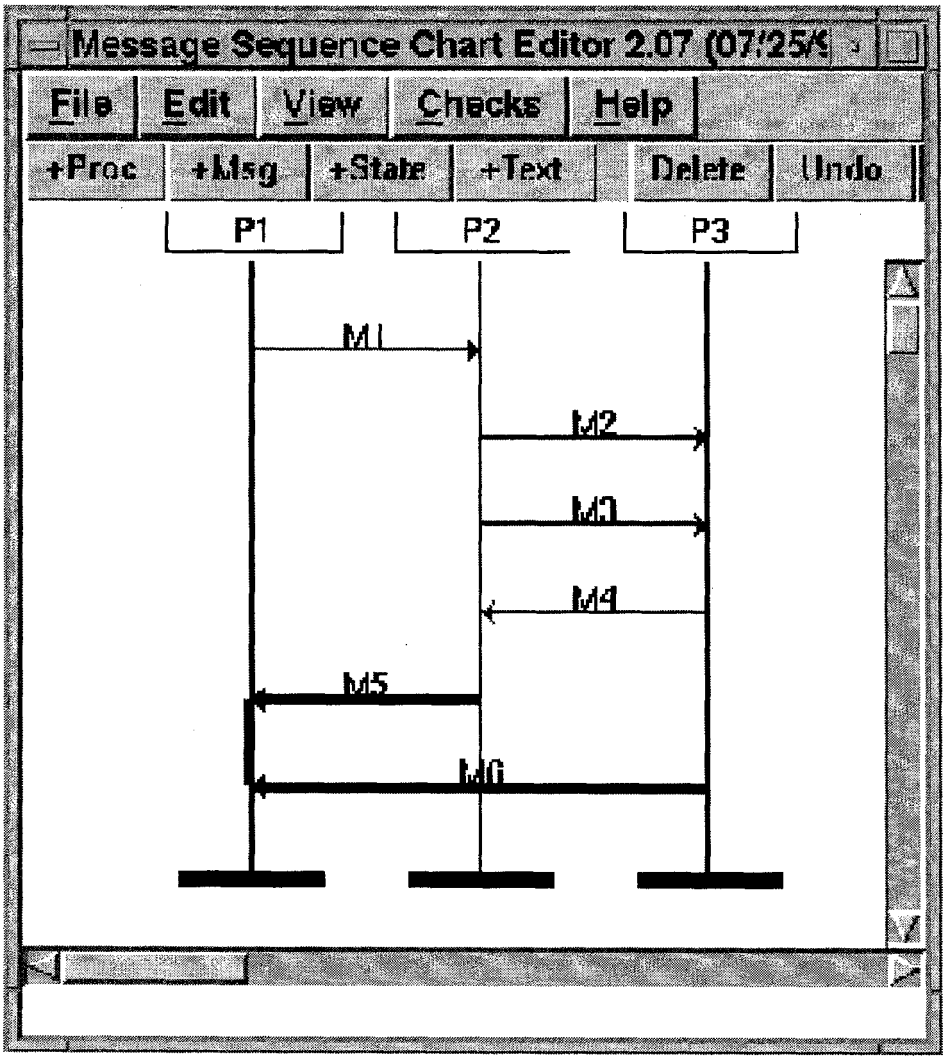


**Fig. 1.** A Message Sequence Chart

advantage that the graphical representation of an MSC is related to some formal representation. We assign to each MSC its semantics denotation as a partially ordered set of *send* and a *receive* event.

The semantics of an MSC depends on some architectural parameters; it can differ between architectures with fifo or non-fifo queues, or between architectures with one or multiple incoming message queue. This results in a slightly different behavior under each such choice. In our implementation, the user is required to select the desired semantics (using a choice menu), and the analysis is done according to this selection. Under fifo with one input queue semantics, the following pairs of events are ordered:

- A send $p$ and its corresponding receive $q$.
- Two sends $p$, $q$ of the same process, where $p$ appears above $q$ on the process line.
- A receive $p$ and a send $q$ of the same process, where $p$ appears above $q$ on the process line.
- Two receives $p$, $q$ of the same process corresponding to sends from some mutual process, where the send corresponding to $p$ appears above the send corresponding to $q$.

Notice that in general, two receives $p$ and $q$ of the same process of messages corresponding to sends from different processes, are not ordered. Similarly, a send $p$ that appears above a receive $q$ in the same process line is unordered with $q$.

We apply some simple verification algorithms to MSCs. One such check is whether the MSCs contain *race conditions*. A race condition can result in from the fact that in most cases, time progresses within each MSC process line downwards; however, this is not always the case, due to some limited control we have over concurrently executed events. For example, the MSC in Figure 1 contains two receive events of process $P1$ (of messages $M5$ and $M6$). Since each process line is one dimensional, the MSC notation forces choosing one of the receive events to appear above the other. However, these two messages were sent from different processes, $P2$ and $P3$, and the chart imposes no order between these sending events. Thus, there is no reason to believe that these messages would indeed arrive in the particular order depicted using the MSC. In Figure 1, the lines corresponding to the two messages in that race appear emphasized. This is a result of applying the race detection algorithm.

Finding races is done as follows [1]: we translate the MSC syntax into its semantical representation as a partially ordered set of events. We calculate the transitive closure $\sqsubset$ of this order using the Floyed-Warshall algorithm [9]. Another order that is constructed from the syntax of the MSC is the *visual order* $<$. This includes pairs of events $p < q$, where $p$ is a send and $q$ is a receive, or $p$ appears above $q$ on a process line. Then a race is reported for each pair of events where $p < q$ but it is not the case that $p \sqsubset q$. In our example, the two receive events $p$, $q$ of messages $M5$ and $M6$, we have $p < q$, since $p$ appears above $q$ on process $P1$ line. But it does not hold that $p \sqsubset q$.

In a similar way, the tool allows checking for time inconsistencies. One can assign lower and upper time limits to intervals between send and receive events. The consistency between these timing constraints are checked and inconsistencies are reported [1].

The MSC standard allows combining simple MSCs using *hierarchical message sequence charts*. These are graphs, where each node is a single MSC. Each path of this graph, starting from some designated initial state, corresponds to a simple concatenation of the MSCs that appear on it. The POGA tool [3] allows to design hierarchical MSCs. With this capability, one can describe a large or even infinite

set of scenarios, each one of them can be finite or infinite (due to loops in the hierarchical graph). The tools MSC and POGA together allow for the creation, debugging, organization, and maintenance of systems of message sequence charts.

The tool TEMPLE [6, 7] adds the ability of searching a hierarchical MSC design for a paths that match a given specification. The specification is also written as an MSC or a hierarchical, using the MSC or Poga tools. The specification MSC is called a *template* and denotes a set of events (sending and receiving of messages) and their relative order. A specification *matches* any scenario that contains at least those events that appear in the template, while preserving the 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, TEMPLE either provides a matching scenario, or the fact that no matching scenario exists in the checked hierarchical graph.

The use of template matching allows one to mechanize such searches. 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 2. 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 with $\sigma_1$, $r_2$ with $\rho_1$, $s_1$ with $\sigma_3$, and $r_1$ with $\rho_3$.

The user interface for MSC and POGA was written in TCL/Tk, and the algorithms were implemented in the language C. The implementation of the template matching was done by translating the hierarchical graph and the template into two COSPAN [5] processes (COSPAN is an automata-based model-checking tool). A third process represents some collected matching information. Then, the automata intersection is performed. If the intersection is not empty, the resulting counter example is translated back into an MSC and displayed. A new implementation was recently written directly in the language SML.

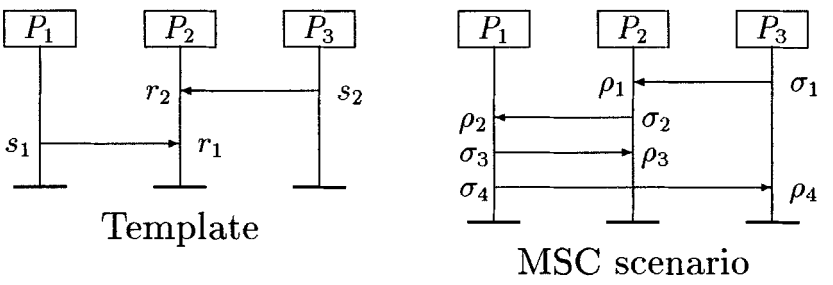


**Fig. 2.** A template and a matching scenario

## Acknowledgements

The tools described in this short paper are a result of collaborations with Rajeev Alur, Brian Kernighan, Gerard Holzmann, Bob Kurshan, Vladimir Levin, Anca Muscholl and Zhendong Su.

## References

1. R. Alur, G.J. Holzmann, D. Peled. An Analyzer for Message Sequence Charts. *Software Concepts and Tools*, Vol. 17, No. 2, pp. 70–77, 1996.
2. A. Ek, J. Grabowski, D. Hogrefer, R. Jerome, B. Kosh, M. Schmitt, SDL'97, Time for testing: SDL, MSC and Trends, Proceedings of the 8th DSL Forum, Elsevier, France, 23–26, 1997.
3. G.J. Holzmann, Early Fault Detection Tools, *Software Concepts and Tools*, Vol. 17, No. 2, pp. 63–69, 1996.
4. ITU-T Recommendation Z.120, Message Sequence Chart (MSC), March 1993.
5. R.P. Kurshan, *Computer-Aided Verification*, Princeton University Press, 1994.
6. V. Levin, D. Peled. Verification of Message Sequence Charts via Template Matching, *TAPSOFT (FASE)'97, Theory and Practice of Software Development*, Lille, France. *Lecture Notes in Computer Science*. Springer, 1997.
7. A. Muscholl, D. Peled, Z. Su, Deciding Properties for Message Sequence Charts, *FoSSaCS, Foundations of Software Science and Computation Structures*, Lisbon, Portugal.
8. B. Selic, G. Gullekson, P.T. Ward, Real-Time Object-Oriented Modeling, Wiley, 1994.
9. S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, **9** (1962), pp. 11-12.