First publ. in: Formale Methoden für verteilte Systeme / Hartmut König (Hrsg.). - München [u.a.], 1993. - S. 88-104. - (Fokus ; 8) - ISBN 3-598-22409-5

On the Semantics of Message Sequence Charts

Peter B. Ladkin
Department of Computing Science
University of Stirling
Stirling FK9 4LA, Scotland
ladkin@compsci.stirling.ac.uk
(JANET: ladkin@uk.ac.stir.cs)

Stefan Leue
Institute for Informatics
University of Berne
Länggassstrasse 51
CH-3012 Berne, Switzerland
leue@iam.unibe.ch

Abstract

We give an informal exposition of a finite-state semantics for Message Sequence Chart (MSC) specifications. We interpret each MSC specification as a transition system of global states, from which we can define a Büchi automaton by considering safety and liveness properties of the system. We show how the set of global states and transitions may be interpreted as a model for temporal logic, and thus how temporal logic formulas may be used to enhance MSC specifications.

1 Introduction

The purpose of this paper is to sketch a proposal for a precise mathematical semantics for Message Sequence Charts, a form of specification method used for telecommunications systems. We will remain informal in our exposition. A formal treatment can be found in [LL92b], [LL92a] and [LL92c].

Message Sequence Charts. Telecommunications protocol specifications are distinguished amongst general system specifications by an emphasis on communication amongst processes rather than computation within a process, and by the relatively simple nature of the messages exchanged. Message Sequence Charts or MSCs (also called *Time Sequence Diagrams* or *Temporal Message Flow Diagrams*) are a form of specification used for telecommunications systems.

MSCs focus entirely on the signals exchanged, and can be a simple, intuitive way of specifying some aspects of telecommunications protocols. They are the subject of an international standardisation effort by CCITT [CCI92]. For use of MSCs in industrial practice, see [AG88], [CCHK90], [CCI88a], and [CCI88b].

Motivation for this work. Our motivation is that the semantics for MSCs are so far relatively undeveloped, in contrast to the syntax definition, which is well developed in [CCI92]. Other papers [Coc91] [CC91] [Til91] focus on formalisations, data structures, and operations on MSCs.

System specification methods used in industry can be very different from those investigated by specification researchers. One might say that common industrial methods are good at bookkeeping, well-engineered, and are relatively easy to train people in, but can be fuzzy in stating system properties. In contrast, logic- or automata-based methods are more precise and expressive, but require greater mathematical or logical skills to use. We believe there is value in bringing the precision of logic-based specification methods to existing industrial methods[1], and this work is a short step in that direction.

Traces are Interleavings. We consider a semantics to be a precise determination of the set of execution *traces* that the specification allows. We use an interleaving model, in which a trace is an interleaving of all observable atomic events of the system which is consistent with the linear ordering of events within each process, from the point of view of a global observer. We note that interleaving models are appropriate for many important specification styles, including TLA [Lam91], CSP [Hoa85] and LOTOS [ISO88][2].

Requirements for the Semantics. We follow two trends in semantics for telecommunications specification. Firstly, we interpret the specification by defining a global system automaton, whose accepted language is identical with the set of system traces allowed by the specification. Secondly, we require that the semantics be finite-state, i.e. that the automaton we define be finite-state. Justification for both these requirements may be found in [Hol91]. The primary reason for the limitations are to ensure that reasonable verification and validation techniques may be applied. There are many practical techniques for analysis and verification of finite-state systems ([CK91] [LS92] as well as [Hol91]). Such methods can exhibit a high degree of automation, and have been used successfully to validate systems with up to 10^{14} states [CD92]. In contrast, non-finite-state methods usually employ theorem-proving techniques which are comparatively human-labor-intensive, are

[1] Rigorous specification methods such as Z and VDM for non-distributed systems are already finding favor in industry. For distributed systems such as telecommunications systems, LOTOS is being developed. These methods have all followed a path from academia to industrial research. In contrast, we are taking a method already in industrial use, and enriching it by providing a more precise and flexible semantics.

[2] Partial order models are often seen as an alternative to interleaving models. We believe much of this contrast is superficial, but it is beyond the scope of this paper to discuss this issue.

often research vehicles, and are currently limited in practical use. It may ultimately be preferable to mix the two sorts of techniques to obtain the advantages of both [Lam92], but as of the time of writing, finite-state methods are the state-of-the-art in telecoms system verification and validation.

Since traces may be infinite, we need a finite-state automaton which accepts infinite strings. The Büchi automaton is probably the most well-known of these, and has been used in the determination of safety and liveness properties of distributed systems [AS87], [AS89]. Given an MSC specification, we define a Büchi automaton and identify the set of system traces specified by the MSC with the set of accepted traces of the automaton.

An alternative interpretation may be considered via Petri Nets [Rud92]. However, this yields a non-finite-state semantics, and thus is unsuitable for our purposes.

The single ne/sig graph arising from an MSC specification may be exponential in the size of the specification, and the global system state graph may be exponential in the size of the ne/sig graph [GHL+92]. But this complexity should be expected. There would be little point in using an MSC style of specification if it were 'just as easy' to write down a global system automaton directly. The specification can be regarded as shorthand for the larger object. For useful specification methods, one should expect that the semantics refers to very large structures, which (one hopes) are still easily analysable and usable without explicit representation.

Overview of the Paper. We first interpret an MSC specification as an *ne/sig* graph, which is a single graph structure representing a set of MSCs. The ne/sig graph is similar to the *flow graphs* of [LS91]. From an ne/sig graph, we then obtain a *global state transition graph*, which is like a finite-state automaton but lacks definition of end-states. We then consider end-state definitions, each of which leads to an automaton. Thus the MSC specification under-defines the resulting automaton. End-state definitions are related to different safety and liveness properties that one might wish to assume as additions to the specification. We show a connection with temporal logic, discuss the formulation of safety and liveness properties in temporal logic, and suggest an enhancement of MSC specifications.

2 From MSCs to ne/sig graphs

We introduce so-called ne/sig graphs (see [LL92b]) as an abstract syntax representation for MSCs. Ne/sig graphs also allow the representation of other message-passing specification methods such as simple SDL (sSDL) [GHL+92] and loop processes [LS91]. This versatility is evidence that they represent a natural syntactical abstraction useful for many different forms of protocol specification. In this paper we will focus on the application of ne/sig graphs to MSC specifications, and we will use them to provide a suitable structure on

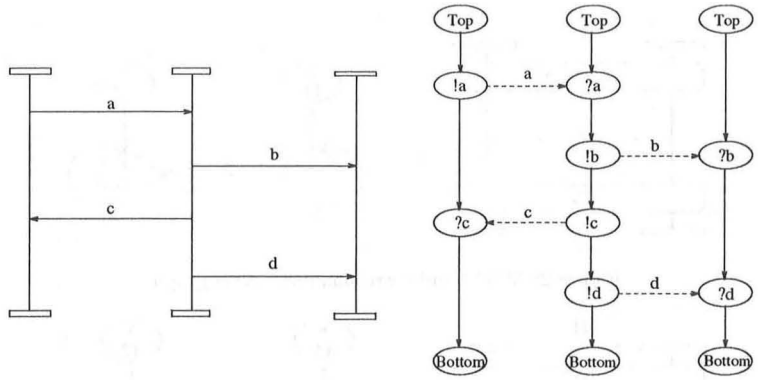

Figure 1: A simple MSC and its corresponding ne/sig graph

which to define states and state transitions.

An MSC Example. Figure 1 shows a *simple* MSC (an MSC without conditions), and its syntactic interpretation as an *ne/sig* graph. A system with three processes is specified. The processes are represented by vertical lines, and the signals sent between processes are represented by horizontal arrows. Communication is asynchronous. The junction between a vertical process line and a horizontal signal line represents an event at which a signal of the type specified is sent or received by the process. In each process axis, the events are temporally ordered from top to bottom. The first process sends a signal of type a to the second process, which upon reception sends a signal of type b to the third process, a signal of type c to the first process, and finally a signal of type d to the third process. The system terminates when all processes have terminated. The ne/sig graph shown on the right is just syntactic sugar, as it is for all simple MSCs. The basic idea is that the events are made explicit as nodes, and the process control-flow edges and signal edges are explicit relations on the nodes.

Ne/sig graphs. We represent an MSC specification (a set of MSCs with conditions) by a single graph with two kinds of edges, *next event* (ne) and *signal* (sig) edges, representing the signals and the progression of processes between events. The nodes represent events, and are labelled with the event type, and the signal edges are labelled with the signal type. The event node at the tail of a sig edge labelled a must be labelled with $!a$ (`send` a message of type a), and the event node at the head with $?a$ (`receive` a message of type a). An ne/sig graph for a simple MSC has extra `start` nodes (in the domain but not the range of the ne relation) labelled *Top*, and `end` nodes (in the range but not the domain

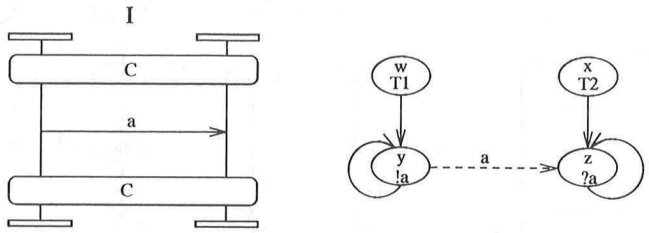

Figure 2: MSC I and corresponding ne/sig graph

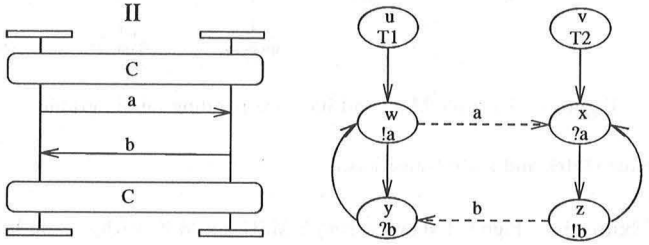

Figure 3: MSC II and corresponding ne/sig graph

of ne) labelled *Bottom*[3].

In different specification styles, communication could be synchronous or asynchronous, channeled or broadcast, finitely or infinitely buffered, reliable or unreliable. Processes may be deterministic or non-deterministic, including parallel constructions or not, terminating or non-terminating. Because a general ne/sig graph may be used to describe a mathematical syntax for different specification methods, the graph is neutral with regard to the meaning of signal edges, the types of communication, or the control structure of processes. However, it does determine statically the sender and recipients of each communication. Thus it is an abstract syntax appropriate for telecommunications specification methods: a *syntax* because it does not encode semantic assumptions, and *abstract* because it abstracts from the details of the syntax of any specific method. The semantics of the specification method arises from how the ne/sig graph is interpreted. For MSCs, we shall interpret it as a particular kind of global state transition system. Hence ne/sig graphs satisfying particular properties represent simple MSCs and MSCs with conditions. We derive from these a single ne/sig graph for an entire MSC specification, which can not in general be represented by a single MSC.

[3]In some ne/sig graph examples in this paper, we also write a lower-case letter in a node to allow us to refer to that node in the text. These letters do not occur in the ne/sig graph itself. The node labels are purely event labels.

Iterations in MSC specifications. Representing entire MSC specifications (which include multiple MSCs) as a single ne/sig graph may require branching or looping in an ne/sig graph, which is disallowed in MSCs. For example, Figures 2 and 3 contain two MSCs with conditions (later called examples I and II, respectively), represented by the elongated symbols labelled `C` spanning the process axes. A condition is like a 'joint' for MSCs. The system is supposed to behave as though another MSC with an identically-labelled condition is joined on at the condition label. In Example I, there is a single condition label `C` at top and bottom. Thus the MSC may be joined to *itself* at these conditions, creating a non-terminating loop in which the first process continuously sends signals of type a to the second. Example II is similar, in which a signals alternate with b signals in the other direction. Both MSCs are represented by ne/sig graphs in which the loops are explicit, as shown.

Non-determinism in MSC specifications Conditions may also be used to specify non-determined behavior, as in Figure 4. We may understand this example as a protocol specification, namely as the connection establishment phase of some very simple connection oriented protocol. When the system is in state `idle`, which means that both processes are in that state, the first process may request the connection establishment by issuing a `CR` request and transit into a local `pending` state. Upon the reception of the `CR` signal the second process transits into its local `pending` state. A *global* state `pending` is reached, if both processes are in the respective local state. As mentioned above we may mark collections of local process states with common labels, in the case of the global system state `idle` with the label `C1` and in case of `pending` with the label `C2`. According to the syntactic definitions in [CCI92], conditions may not cut through message arrows, thus they only represent global system states in which no message is in transit. Conditions only represent *possible* global system states - it is not required that these global system states are ever actually reached during execution of a system. At the condition `C2`, the second process may send a `CC` signal to the first, which indicates a confirmation to the connect request and a transition to the global state `connected`, or alternatively a `DR` signal to signal rejection of the connect request, before looping back to the beginning (condition `C1`). This gives rise to the branching and looping ne/sig graph in Figure 6.

The translation is handled in [LL92c] first by translating the MSCs with conditions into ne/sig graphs with condition nodes (Figure 5), which are an extra kind of node on each process axis, then joining the ne/sig graphs at these nodes and eliminating the condition nodes which are not required to synchronise non-local branching of process control. The formalisation of this *unfolding* is straightforward, but requires care in the details.

Ne/sig Graphs Formally. Formally, an ne/sig graph is a tuple

$$N = (S, C, X, ne, sig, ST, stype, Top, Bottom)$$

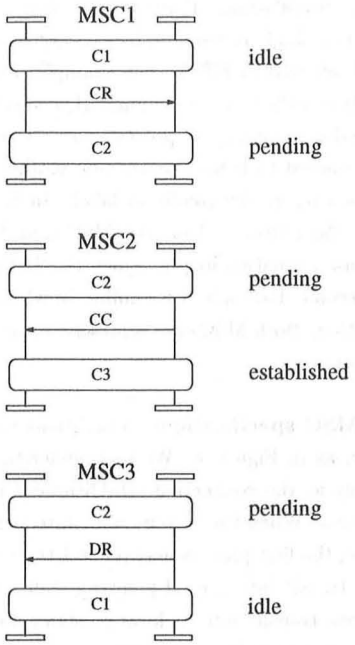

Figure 4: MSC specification with conditions

where S and C are respectively the sets of `send` and `receive` nodes, and X is the set of nodes used for start and end nodes and conditions. S, C and X are pairwise disjoint. ne is the ne relation on $(S \cup C \cup X) \times (S \cup C \cup X)$, and sig is the sig relation on $S \times C$ ST is the set of signal types, $stype$ the labelling function for the sig edges, and Top and $Bottom$ the labels for the start and end nodes.

3 Global State Transition Graphs

The ne/sig graph represents an entire MSC specification by a single graph. In order to obtain an automaton from an ne/sig graph, we first have to define the global states, the start state, and the state transition function. This triple defines the global state transition graph (GSTG), and is uniquely determined by the MSC specification[4]. We require that there must be a finite number of global states.

[4]To make an automaton from the GSTG, we need to define the set of final states

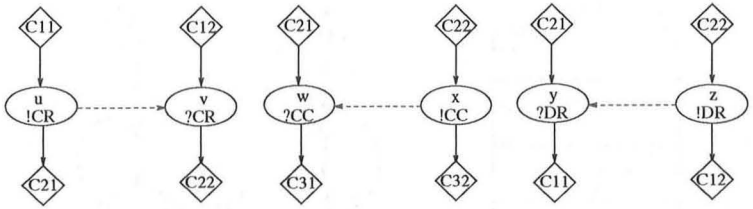

Figure 5: ne/sig graphs with conditions

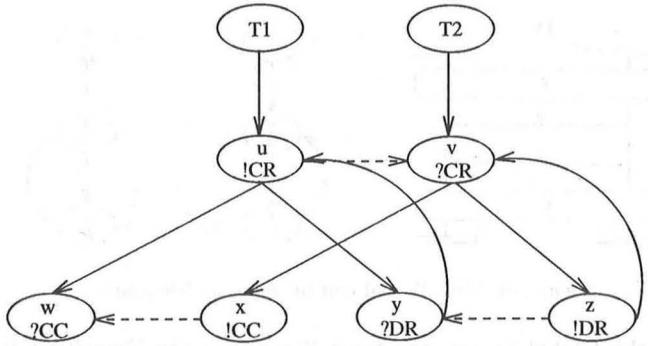

Figure 6: 'Unfolding' an MSC specification into a single ne/sig graph

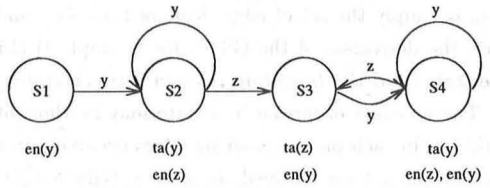

Figure 7: Global State Transition Graph for example I

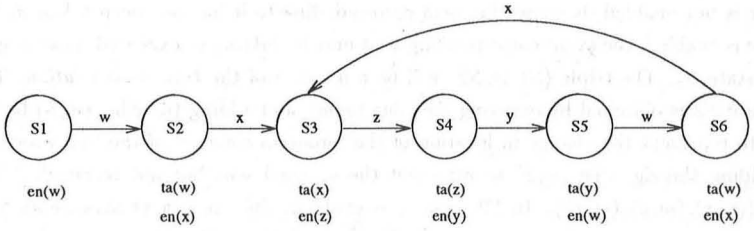

Figure 8: Global State Transition Graph for Example II

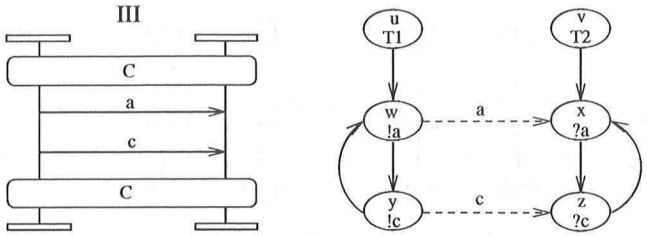

Figure 9: MSC III and corresponding ne/sig graph

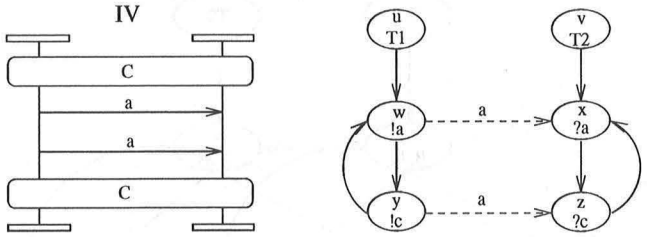

Figure 10: MSC IV and corresponding ne/sig graph

Obtaining the Global States, the Start State, and the Transition Relation. The *global states* are certain sets of edges of the ne/sig graph, and the transition relation between states is obtained by deleting particular edges from the state and adding others. The *start state* q_0 is simply the set of edges leading from *Top* nodes in the graph. We shall walk through the derivation of the GSTG for Example II (Figure 3) given in Figure 8, to illustrate states and the *transition relation* between states. The start state $q_0 = \{(u,w),(v,x)\}$. The ne edges occurring in a state may be thought of as the set of positions where control lies in each process, and sig edges occurring in the state may be thought of as signals sent but not yet received. In state q_0 (labelled $S1$ in Figure 8) the event of type $!a$ at node w is *enabled*, because node w represents a `send` node (a `send` node p is enabled in a state S if there is an ne edge with p as second coordinate in S). Node x is not enabled, because the `send` corresponding to it has not been taken in $S1$. Since w is enabled, the event corresponding to it may be *taken*, i.e. executed, next to give a new state S_2. The triple $\langle S1, w, S2\rangle$ will be a member of the *transition relation*. The new state $S2$ is obtained by omitting the edge (u,w), and adding the edge (w,y) to the state (to represent the change in location of the 'program counter' of the first process), and adding the sig edge $\langle w,x\rangle$ to represent the a signal sent but not received. Thus $S2 = \{(v,x),(w,y),\langle w,x\rangle\}$. In $S2$, node x is enabled, since it is a `receive` node and requires not only that its 'program counter' be at the right position (i.e. an ne edge with x as second coordinate is in the state), but that a sig edge with x as second coordinate is also in the state (i.e. the signal has been sent). When the action corresponding to node x is *taken*, the edges $\langle w,x\rangle$ and (v,x) are removed from the state $S2$, and (x,z) is added to represent advance of the program counter. The resulting state is $S3 = \{(w,y),(x,z)\}$. $\langle S2,x,S3\rangle$ is in the transition relation. Node z is enabled in $S3$, and so on. The GSTG in Figure 8 is annotated with the list of actions enabled (*en()*) and taken(*ta()*) in each state. Figure 7 shows the GSTG for Example I.

MSC Specifications can 'Count' Receptions. As we argued in [LL92c] Example I, Figure 2, and Example IV, Figure 10, do not represent the same set of traces. Both examples express a non-terminating `send` of a signal of type a from the first process to the second. However, Example I allows only *one* `receive` event, which is succeeded by an arbitrary number of `send`s, whereas Example IV allows *two* consecutive `receive`s, but no more, and any number of consecutive `send`s.

This interpretation of Examples I and IV may be counterintuitive. Firstly, why do I and VI not define the same set of traces? Secondly, why reject a trace in which there are, say 123 sends before the first receive? Surely, these should be valid interpretations. Formally interpreted, this suggestion says that the intuitive interpretations are those traces in which at any point in the trace, the number of `send`s is greater than or equal to the number of `receive`s. This could easily be accomplished by simply counting the number of times the edge $\langle y,z\rangle$ is inserted into the edge-set defining the state. However, it is easy to see there would now be a state, for each positive integer n, in which the count of this edge is n. Thus we are no longer finite-state, indeed it is well-known that one cannot devise a finite automaton to accept precisely the suggested set of traces.

There is thus a conflict between naive intuition concerning the particular specifications here, and intuition concerning the finite-state nature of the individual processes. In our view, then, the non-intuitive interpretation of these particular MSC examples is both expected and appropriate.

GSTGs can be Complicated. It should be no surprise that GSTGs can rapidly become very complicated, for example the GSTG for Example III in Figure 9 has 21 states (see [LL92c]). This is partly due to the asynchronous communication, and partly to interleavings of non-related events. Example II and Example III are similar, differing only in that the second message goes in opposite directions. In Example II this forces a unique execution sequence, and the GSTG is correspondingly simple (Figure 8). However, in example III, the two `send`s might occur before either `receive`, or alternatively `send`s and `receive`s might be interleaved. Thus the GSTG is more complex. However, it is not our intention to recommend explicit construction of the GSTG for every MSC, for the usual state-explosion reasons advanced in [Hol91]. We use it later formally to relate liveness and safety properties as expressed in temporal logic or by Büchi automata to MSCs.

4 From GSTGs to Automata

The global state transition graph, which we defined in the previous section, is almost an automaton, lacking only a definition of end states, to which we now turn.

Definition of global state automaton. Let M denote a MSC specification and $\mathcal{GSTG}_M = (Q, q_0, T_M)$ the corresponding global state transition graph, where Q denotes the set of reachable global system states, q_0 the initial global system state and T_M the transition relation[5]. We can define a Büchi automaton which transits between global system states, by adding to $\mathcal{GSTG}_M$ a definition of a set of *final states* F. The definition of a Büchi automaton is very similar to that of the usual finite-state automaton, except for the criterion for *acceptance* of a string. Büchi automata may accept infinite strings. A *global state automaton for* $\mathcal{GSTG}_M$ is $A_M \triangleq (Q, q_0, T_M, F)$, where $F \subseteq Q$ is a set of *final states*. Acceptance is Büchi-acceptance [Tho90], namely an infinite word is accepted iff the automaton cycles through some state in F infinitely often on the word (the alphabet is the set of events, e.g. $?a, !b$, and a word is thus a possibly infinite sequence of events, i.e. a possible trace).

Assume that the global state transition graph with 3 global states in Figure 11 is derived from some MSC specification, and $q_0 = S1$. The set of infinite paths through the graph is represented by the ω-regular expression

$$(!a(!b?b)^\omega) + (!a(!b?b)^*?a)^\omega + (!a(!b?b)^*?a)^*.(!a(!b?b)^\omega).$$

Selecting $F = \{S2, S3\}$ as end-states means that traces of the form $!a(!b?b)^\omega$ would be accepted. Traces in this class do not satisfy the liveness requirement that a sent message will eventually be received (the counter example here is $!a$ in the first and third terms in the sum). However, selecting $F = \{S1, S2\}$ ensures that only the *fair* traces of the form $(!a(!b?b)^*?a)^\omega$ are accepted. Thus selection of a set of end states depends fundamentally on the liveness and safety characteristics assumed for a particular MSC specification. [LL92c] contains examples of liveness properties and their relation to end-state selection.

5 MSCs and their connection to Temporal Logic

In the last section we discussed liveness properties for MSC specifications. A discussion of the use of Büchi automata to specify such properties of distributed systems can be found in [AS87] and [AS89]. A complementary approach for expressing safety and liveness properties may be found in the use of temporal logic. Temporal logic formulae are interpreted over infinite sequences of states, each state being defined by the truth values of state predicates. We relate these formulae to the automata obtained from the semantics

[5]For a formal definition of GSTGs see [LL92c].

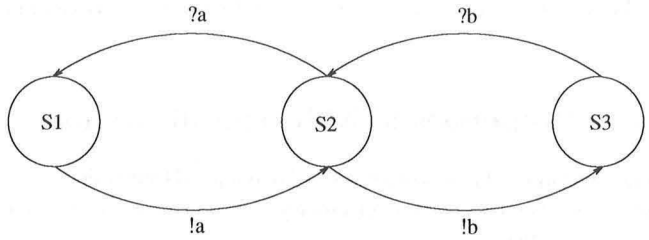

Figure 11: Global state transition graph

definition. We remain informal here, referring the reader to [LL92c] for more precision. We base our temporal logic interpretation on the Manna-Pnueli approach [MP92].

Basic transition systems Following [MP92] we interpret global state transition graphs as so-called *basic transition systems* (BTS). A BTS consists of a *finite set of states* Σ, a *transition function* τ mapping a state to a set of possible successor states, and an *initial condition*. We denote the set of all transitions τ by T. For an MSC M, Σ will be the set of states Q of $GSTG_M$, the transitions τ will be the communication events that lead from one global state to another, and the initial state of the BTS will be the initial state of the GSTG.

Computations and State Predicates Manna and Pnueli define the following notions [MP92]. An infinite state sequence $\sigma = s_0, s_1, \ldots$ is a *computation* iff s_0 is the initial state of the BTS, and for all consecutive pairs $s_i, s_{i+1} \in \sigma$ there exists $\tau \in T$ such that $s_{i+1} \in \tau(s_i)$. The indices i of σ are *positions*. Transition τ is *enabled at position* i of some computation σ, written as $en(\tau)$, iff $\tau(s_i) \neq \emptyset$. Transition τ is *taken* at position $i+1$, written as $ta(\tau)$, iff $s_{i+1} \in \tau(s_i)$.

To correlate these definitions with the global state transition graph, we need to define the *enabled* and *taken* predicates. Roughly speaking, a transition is *enabled* if it is enabled in the sense used earlier in Section 3. Similarly, a transition is *taken* in a state if that transition has led to the state from an immediately preceding state (notice the 'past tense' sense of the predicate *taken*). Details may be found in [LL92c].

Temporal Logic. Given these interpretations of a GSTG as a model for temporal logic, we may define a temporal logic in the usual way, e.g. [MP92]. The language has state predicates $en(\tau)$ and $ta(\tau)$ as only basic propositions, includes the Boolean connectives (we use just $\neg$ and $\vee$ for simplicity), and the temporal operators $\Diamond$ (eventually), $\Box$ (henceforth), $\Diamond\!\!\!-$ (sometime in the past), $\ominus$ (previous) and S (since). The semantics are defined as usual. A temporal logic formula p is interpreted over state sequences σ, and we

define the usual model-theoretic notion $(\sigma, i) \models p$, that formula p is *satisfied* in position i of sequence σ.

6 Logical Properties of MSC specifications.

We can now give examples of properties expressed in temporal logic which can characterize MSC specifications. The classification of properties as *safety, recurrence*, etc, refers to the classification in [MP90].

Properties Satisfied by all MSC Specifications. The following properties are satisfied by all computations derived from MSC specifications, as may be seen by inspection.

1. **Enabling of a `send` event (a *safety* property):** If a `send` event is taken, it must have been enabled previously. However, the enabling does not have to persist until the event is disabled, because a `send` event may also be disabled by a nondeterministic behavior alternative (some branch is taken rather than another) in the process's control flow.

   $$\Box(ta(x) \supset \Diamond\!\!\!-\, en(y))$$

   where $\langle x, y \rangle \in sig$.

2. **Persistent enabling of a `receive` event: (a *safety* property)** A `receive` event may only be taken if it has been previously enabled by a `send` event of the same type. Additionally, the enabling of a `receive` event can only be disabled by a `receive` event, therefore an enabling of a `receive` event persists up until the state when it is taken.

   $$\Box(ta(y) \supset \ominus\, (en(y)\ S\ ta(x)))$$

   where $\langle x, y \rangle \in sig$.

Some Potential Requirements on MSC Specifications. Some liveness properties are not automatically fulfilled by an MSC specification M. It was noted earlier that some of these properties were definable by making different selections of the set of final states of a Büchi automaton defined on $GSTG_M$. Therefore, the MSC specification method may be enhanced by requiring explicit statements of which liveness properties are to be satisfied in a given specification. Well-known examples of such properties are

1. **Weak fairness (a *recurrence* property):** it is not the case that any transition τ is enabled continuously without ever being taken.

   $$\Box\Diamond(\neg en(\tau) \vee ta(\tau))$$

2. **Strong fairness (a *reactivity* property):** if an arbitrary transition τ is enabled infinitely many times, then it is taken infinitely many times.

   $$\Box\Diamond en(\tau) \supset \Box\Diamond ta(\tau)$$

It is known (and should be clear) that strong fairness implies weak fairness. We note that since `receive` events are persistently enabled, strong fairness and weak fairness just for `receive` events are equivalent statements. However, since a `send` event may be disabled without being taken, strong fairness and weak fairness are not equivalent for `send` events.

A Proposal for Enhancing MSC Specifications. By means of our explicit connection between MSC specifications and temporal logic semantics, the imprecision in MSC specifications resulting from the lack of specification of liveness or fairness properties can be remedied, if desired, by allowing explicit statements in temporal logic of such properties as part of the MSC specification.

7 Concluding remarks

We have provided a finite-state semantics for MSCs. We first interpreted an MSC specification as a single graph structure, the ne/sig graph, and then defined a collection of global system states, and transitions between them, from this graph. We then discussed the completion of this graph to a Büchi automaton, noting the reliance on liveness properties which are not explicit in the MSC specification, and made an explicit connection to temporal logic via the standard semantics. This connects MSCs with the most rigorous specification methods available for distributed systems, and allows MSC practitioners to make use of the considerable literature on these methods. Consequently, we noted that temporal logic formulae may be used to enhance MSC specifications, if so desired.

Acknowledgements

We sincerely thank Dr. Ekkart Rudolph, Rapporteur of the CCITT Z.120 Working Group on Message Sequence Charts, Professor Ken Turner and Philippe Oechslin for their careful reading and helpful commentary on drafts of this paper.

Both authors started looking at MSCs under contract 233 of the Swiss PTT to the University of Berne, Project Head Professor Dieter Hogrefe. During this time, we benefitted particularly from discussions with co-workers Jens Grabowski and Robert Nahm.

The first author was supported by IBM Almaden Research Center and the CEC RACE II R2088 project TOPIC. The second author was partly supported by the Swiss National Fund.

References

[AG88] Siemens AG. *EWSD Softwareentwicklungshandbuch (Software Development Handbook), Kapitel B, Register 6, SDL Diagramme.* Siemens AG, München (Munich), 1988.

[AS87] B. Alpern and F.B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

[AS89] B. Alpern and F.B. Schneider. Verifying temporal properties without temporal logic. *ACM Transactions on Programming Languages*, 11(1):147–167, 1989.

[CC91] A. Cockburn and W. Citrin. An executable specification language for history-sensitive systems. Technical Report IBM RZ 2162, IBM Rüschlikon Research Laboratory, Zürich, 1991.

[CCHK90] A.A.R. Cockburn, W. Citrin, R.F. Hauser, and J. Känel. An environment for interactive design of communication architectures. In *Proceedings of the IFIP symposium on Protocol Specification, Testing and Verification, X*, 1990.

[CCI88a] CCITT. Recommendation Q.65: Stage 2 of the method for the characterization of services supported by ISDN. CCITT, Geneva, 1988.

[CCI88b] CCITT. Recommendation Q.699: Interworking between the digital subscriber system layer 3 protocol and the signaling system no. 7, ISDN user part. CCITT, Geneva, 1988.

[CCI92] CCITT. Recommendation Z.120: Message Sequence Chart (MSC). CCITT, Geneva, 1992.

[CD92] D. Cohen and N. Dorn. An experiment in analysing switch recovery procedures. In *Participants Proceedings of FORTE'92: Formal Description Techniques*, pages 17–25, Oct 1992.

[CK91] E.M. Clarke and R.P. Kurshan, editors. *Computer Aided Verification: Proceedings of CAV'90*, volume 531 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.

[Coc91] A.A.R. Cockburn. A formalization of temporal message-flow diagrams. In *Proceedings of the IFIP Symposium on Protocol Specification, Testing and Verification, XI*, 1991.

[GHL+92] J. Grabowski, D. Hogrefe, P. Ladkin, S. Leue, and R. Nahm. Conformance testing - a tool for the generation of test cases. Project Report, project contract no. 233, funded by Swiss PTT, University of Berne, May 1992.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[Hol91] G. J. Holzman. *Design and Validation of Computer Protocols.* Prentice-Hall International, 1991.

[ISO88] ISO. Information processing systems - open systems interconnection - LOTOS : A formal description technique based on temporal ordering of observed behavior, international standard 8807. International Standards Organisation, 1988.

[Lam91] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, Dec 1991.

[Lam92] L. Lamport. Applying finite-state methods to infinite-state systems. *TLA Mailing List Note*, 92-08-28, 1992.

[LL92a] P. B. Ladkin and S. Leue. An analysis of message sequence charts. Technical Report IAM 92-013, University of Berne, Institute for Informatics and Applied Mathematics, 1992.

[LL92b] P. B. Ladkin and S. Leue. An automaton interpretation of message sequence charts. Technical Report IAM 92-012, University of Berne, Institute for Informatics and Applied Mathematics, 1992.

[LL92c] P. B. Ladkin and S. Leue. Interpreting message sequence charts. Technical Report IBM RJ 8965, IBM Almaden Research Center, San Jose, CA, 1992.

[LS91] P.B. Ladkin and B.B. Simons. Compile time analysis of communicating loop processes. Technical Report IBM RJ 8488, IBM Almaden Research Center, San Jose, CA, Nov 1991.

[LS92] K.G. Larsen and A. Skou, editors. *Computer Aided Verification: Proceedings of CAV'91*, volume 575 of *Lecture Notes in Computer Science.* Springer Verlag, 1992.

[MP90] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 377–408. ACM Press, Aug 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

[Rud92] E. Rudolph. personal communication, 1992.

[Tho90] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, chapter 4, pages 132–191. Elsevier Science Publishers B. V. (North-Holland), 1990.

[Til91] P. A. J. Tilanus. A formalisation of message sequence charts. In O. Faergemand and R. Reed, editors, *SDL '91: Evolving Methods*, pages 273–288. Elsevier Science Publishers B. V. (North-Holland), 1991.