What Do Message Sequence Charts Mean?

Peter B. Ladkin[a*] and Stefan Leue[b†]

[a] Department of Computing Science, University of Stirling, Stirling FK9 4LA, Scotland

[b] Institute for Informatics and Applied Mathematics,
University of Berne, Länggassstr. 51, CH-3012 Berne, Switzerland

Abstract

We propose a semantics for Message Sequence Charts (MSCs). Our requirements are: to determine unambiguously which execution traces are allowed by an MSC; and to use a finite-state interpretation. Our semantics handles both synchronous and asynchronous communication. We define a global state automaton from an MSC, by first defining a transition system of global states, and from that a Büchi automaton. In using MSCs, properties of the environment and liveness properties of the MSC itself may be under-specified. We propose a method using temporal logic formulas to specify the required liveness properties.

Keyword Codes: F.3.2; D.2.1; 2.
Keywords: Semantics of Programming Languages; Requirements/Specifications; Protocol specification, testing and verification.

1. INTRODUCTION

The purpose of this paper is to to propose a precise semantics for Message Sequence Charts (MSCs) [16]. Our interpretation also suffices to give a semantics for Time Sequence Diagrams [34]. We explain our semantics intuitively, and discuss some issues which arise. Mathematical details may be found in the full report [35].

Message Sequence Charts. Telecommunications protocol specifications are distinguished by an emphasis on communication between processes rather than computation within a process, and by the relatively simple nature of the messages exchanged. Message Sequence Charts (MSCs) and their equivalents are used both formally and informally to describe fragments of message-passing behavior in telecommunications systems, networks in general, or even in object-oriented analysis [19] ('temporal message flow diagrams'), [44] (many examples) [47] ('primitive sequences') [43] ('event traces') [1] [20] [13] [14]. MSCs are subject to standardization by CCITT [16].

Figure 1 shows a simple MSC. Processes are represented by vertical lines with time progressing downwards. Messages exchanged between processes are represented by horizontal or sloping directed lines.

*Work supported by IBM Almaden Research Center and the CEC RACE II project R2088 TOPIC.
†Work supported by the Swiss National Science Foundation and the Swiss PTT.

Motivation for this Work. A precise semantics for any specification aid is a necessity. However, the semantics for MSCs are relatively undeveloped, in contrast to the syntax [16]. Other papers [18] [46] focus on formalizations, data structures, and operations. [11] mentions MSCs but uses them only to define traces of SDL specifications. A process algebra-based approach for semantics has been suggested [12], [41] but as yet it handles only synchronous communication, whereas asynchronous message-passing is more common. A Petri-net based approach has also been suggested [26].

Traces are Interleavings. We consider a semantics to be a precise determination of which execution *traces* the specification allows. MSCs focus on communication events hence we represent only these in a trace: internal process computation is ignored. In protocol conformance testing, the inputs and observations of a test are displayed as a linear sequence of signals. This suggests that for us a trace should be an interleaving of observable atomic events, as in TLA [37] or CSP [27].

Justification for our Semantics. Most methods of analysing telecommunications system specifications calculate and analyse the global system states [29] [17] [38]. This requires that there should only be finitely many states. Such methods have been used successfully to validate systems with up to 10^{14} states [21]. Since traces may be infinite, we need a finite-state automaton which accepts infinite sequences. Büchi automata have been used in the determination of safety and liveness properties of distributed systems [4], [5]. A Büchi automaton is similar to a normal finite-state machine, but it may accept infinite sequences, using the criterion that an infinite sequence is accepted if the automaton passes through a final state unboundedly often on the sequence. We shall identify the set of traces specified by an MSC with the set of accepted traces of a Büchi automaton constructed from the MSC.

Complexity. The single graph structure that we calculate 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 this graph, giving a crude doubly-exponential estimate of the bound on the size of the automaton. But this explosion should be expected. There would be no point in using MSCs if it were just as easy to write down a global system automaton directly. The MSC can be regarded as shorthand for the larger object, which should nevertheless permit analysis without explicit representation.

Handling Synchronous Communication. The constructions in this paper also handle synchronous communication, including mixtures of synchrony and asynchrony, with minor modification. We illustrate constructions for asynchrony, and indicate how definitions of enabled event and state transition are modified for synchrony. But why include synchrony? Firstly, some arguments [27] [42] suggest that effective formal methods are easier to devise for synchronous primitives. Secondly, many relevant languages rely on synchronous primitives, e.g. *CSP* [27], *CCS* [42], *LOTOS* [33], *OCCAM* [30], *ESTEREL* [9], [10], *SIGNAL* [8, 7] and *LUSTRE* [6]. Thirdly, in some languages synchrony and asynchrony co-exist, e.g. the dialect *ESTELLE** [22, 23] of *ESTELLE* [32]; and a suggested extension of SDL [28]. For such languages to avail themselves of MSCs, synchronous primitives must be handled. Lastly, in ISO OSI [31], synchronous events naturally occur at the interface between layers, because the same event is looked at in two different ways by the two different layers.

Requiring a Finite-State Semantics. We have already noted that current verification methods mostly require finite-state semantics. Another reason for a finite-state semantics arises from considering what system information is explicit, and which hidden, in the specification method. We argue that the explicit information available from an MSC allows only finitely many global system control states.

The argument proceeds from considerations of fault tolerance. Each individual process control is a finite-state device with respect to `send`s and `receive`s. A finite state device can only remember a bounded computation history. Suppose, as the system runs, communication channels are compromised (for example, someone cuts a cable). Also assume the processes themselves are not compromised. A consistent state must be reconstructed. Each process should know its state, namely, where it thinks its control is and what it remembers from its history. No other information may be assumed to be available. What can be reconstructed from this information is thus bounded, no matter how long the system has been running previous to the fault. Hence, two failures which result in the same local states of the processes are equivalent from the point of view of the reconstructable state of the system. Each such equivalence class can be identified with a *global state*. Since there are finitely many finite-state processes, the global states are some equivalence (probably the identity) relation on a subset of the cartesian product of the state spaces of the individual processes, and thus there are only finitely many global states. A conservative upper bound to the number of these states is the size of this cartesian product.

Overview of the Paper. We first relate the MSCs in this paper to MSCs in the CCITT standardized recommendation. Then we list and discuss some MSC tools. Next, we interpret an MSC specification (a set of MSCs) as an *ne/sig* graph, a structure similar to the *message flow graphs* of [36]. From an ne/sig graph we obtain a *global state transition system*, which is like a finite-state automaton but lacks definition of end-states. Considering end-state definitions leads to a Büchi automaton. The MSC specification under-defines the automaton, in that end-state definitions are related to different safety and liveness properties not explicit in the MSCs. We show how these properties may be defined via a connection with temporal logic.

2. MSCS IN THIS PAPER VS. PROPOSED STANDARD MSCS

The standardization effort for MSCs is an activity of the SDL standardization group. We note here some differences between our notation, and the proposed MSC standard, *Recommendation Z.120, Message Sequence Chart* [16]. There is no formal semantics described in Z.120, but some informal comments are given. Our comments here refer to these informal explanations.

MSCs and SDL. Although MSCs are often used in combination with SDL [15], similar devices are used in other contexts also, as we have noted. So, unlike Z.120, we do not define an MSC to represent a set of traces of an SDL-specification. An MSC for us more generally defines a set of traces of `send` and `receive` events of typed messages.

Environment. Z.120 treats the behavior of the environment in a particular way, namely events in the environment have an arbitrary ordering. We do not treat communication with the environment in such a special manner, but our semantics may easily be modified to do so if required. Further, the environment in Z.120 can have implicit, and sometimes counterintuitive, properties, and we believe it is false to claim that the environment can always be represented explicitly by an additional process-like axis, as suggested in Z.120 (see later).

Conditions. We only consider global initial and final conditions, and have some reservations even about these [35].

Abstraction. We do not consider refinement and abstraction corresponding to the *process* and *sub-MSC* concepts of Z.120.

Process Model. The process model discussed in this paper does not allow for dynamic generation of processes. However, it may easily be modified to include dynamic process generation, by means of `create` nodes in the ne/sig graph with similar semantics to that of a synchronous communication, with the constraint that only a `Top` node may precede a `create` receive event (in the created process).

Coregion. We have no concept comparable to the *coregion* of Z.120. Coregions could be treated similarly to event orderings for the environment.

Global System States. According to Z.120 '*a global system state is determined by the values of the variables and the state of execution of each process and the contents of the message queues*'. This seems to contradict the informal semantics of Z.120, which contains no concept of data and thus no concept of variables. Furthermore, message queues are not represented explicitly in MSCs, and we have already presented an argument from fault-tolerance why this should remain so.

There is a further human-factors argument against incorporating message queues. Users of MSCs rarely think of the states of queues in order to see what their MSCs define. The appeal of MSCs lies in their relatively simple graphics for talking about sequences of messages. If determining the global system state required information about the contents of queues at any point in a trace, then 'what you see' in an MSC would not be 'what you get', i.e. there would be non-explicit semantic information concerning the history of a computation that had to be taken into account when determining the next state of the system. We believe it is a bad idea in general to employ specification methods with hidden elements (e.g. properties of the environment, states of buffers or queues).

3. MSC TOOLS

Various MSC-based tools have been implemented, for example by Siemens AG (ZFE Division in München, Germany), AT&T Bell Labs (Naperville, Illinois, USA), University of Berne (Switzerland), Telelogic (Malmö, Sweden), and Verilog (Toulouse, France). Telelogic's SDT tool as well as Verilog's GEODE SDL tool are sold commercially. The test generation tool from the University of Berne [25] is based on our semantics.

Comparison with the GEODE Toolset. Since the requirements for MSC semantics in GEODE are similar to ours, we describe how MSCs are used in GEODE. GEODE's MSC tools are being enhanced within the AVALON project [3]. MSCs are used as a special kind of observer dealing with signal sequences, to specify parts of traces which may or may not be explicit in a given SDL specification. One may compare traces defined by MSCs with SDL specifications in GEODE in a variety of ways.

GEODE employs two styles of interpretation, *local ordering*, which only considers ordering of events relative to a given process, and looks at the event ordering of each process independently; and *global ordering*, in which an event occurs before another if and only if that event occurs graphically higher up in the entire MSC diagram. A given MSC defines a unique trace under global ordering.

Our MSC semantics produces a finite-state automaton from what [3] calls the *causal ordering*. The causal ordering is favored by Z.120. GEODE does not implement it, because under this ordering MSCs "cannot be formalized easily as automata. This makes it difficult to use causal ordering [along with other GEODE methods]" [3]. Our work formulates the causal ordering using finite state automata, providing a solution to this (supposed) problem within GEODE.

Other operators on MSCs are available in GEODE, and are being extended in AVALON, such as *sequencing* (our composition), *exclusion*, *exception* and *loops* (which appear in our ne/sig graphs). These operators on MSCs are transformed into operators on the FSMs that interpret the global or local semantics, so that a global MSC may be obtained. In general, this global MSC is non-deterministic, and is transformed into a deterministic machine for the validation tools [2].

There are some further questions about global orderings. What is the interpretation of an MSC if both the `send` and the `receive` events are located at the same height in the MSC diagram? And what is the interpretation if the `send` event occurs further down than the `receive` event to which it is related by the message arrow? There may be some pragmatic answer inside the GEODE tool set. However, such questions require a principled answer, which our semantics provides[3].

4. FROM MSCS TO NE/SIG GRAPHS

We introduce *ne/sig graphs* as an abstract syntactic representation for MSCs. Ne/sig graphs are a natural syntactical abstraction useful also for other forms of specification, such as simple SDL (sSDL) [24], Time Sequence Diagrams, and loop processes [36].

In this section, we show how to derive ne/sig graphs from MSC specifications. An *MSC specification* is a set of MSCs with conditions (see Figure 3).

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 for simple MSCs is just syntactic sugar. 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. In order to handle both synchronous and asynchronous communication, two different types of signal edges are required. Thus there are two signal relations on nodes of the graph. The state

[3]In particular our translation of the graphical object *MSC* into an algebraic object *ne/sig graph* avoids similar ambiguities (see Section 4). We define the *ne* and *sig* relations to imply an ordering of `send` and `receive` events which is independent of their location in the MSC diagram.

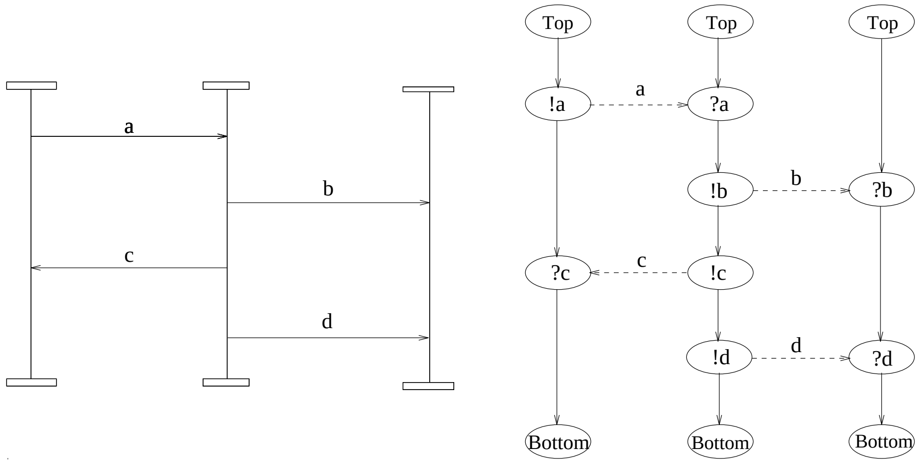

Figure 1. A simple MSC and its corresponding ne/sig graph

transition relation on these two types of nodes will be treated somewhat differently, and the definition of when an action is enabled will differ on synchronous and asynchronous actions.

Ne/sig Graphs. Ne/sig graphs have 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 labeled with the event type, and the signal edges are labeled with the signal type. The event node at the tail of a sig edge labeled a must be labeled 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 has `start` nodes (in the domain but not the range of the ne relation) labeled *Top*, and maybe `end` nodes (in the range but not the domain of ne) labeled *Bottom*[4].

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, Figure 2 contains two MSCs (MSCs I and II) with conditions, represented by the elongated symbols labeled `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-labeled condition is joined on at the condition label. In MSC 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. MSC 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 3. We may understand this example as a protocol specification, namely as the connection establishment phase of some very simple

[4]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.

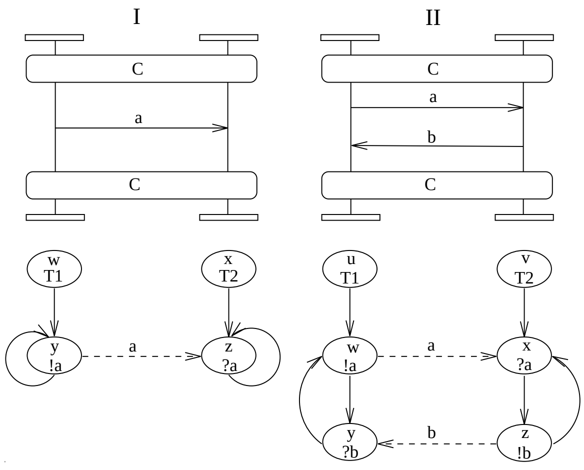

Figure 2. MSCs I and II and corresponding ne/sig graphs

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 pending 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 [16] 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 4.

The translation is handled 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 finally eliminating the condition nodes. The formalization of this *unfolding* is straightforward, but requires care. The technical details are in [35].

However, as we show in [35], unrestricted use of conditions, even of this simple form, leads directly to the need for state history predicates whose values need to be known by more than one process. To us, it would be defeating the purpose of MSCs that the environment could retain such state history and communicate it to each process partner, without this communication being represented explicitly in the MSC. Thus, some means must be found to restrict the use of conditions, but it is beyond the scope of this paper to suggest how.

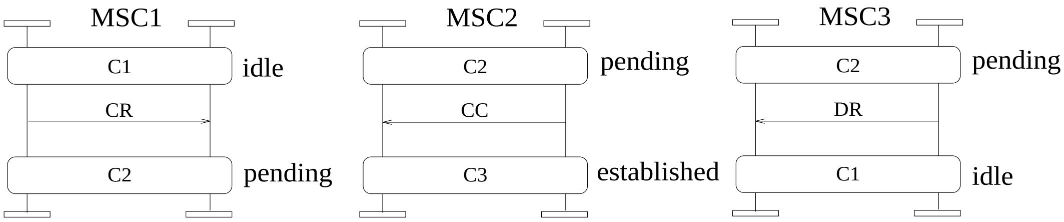

Figure 3. MSC specification with conditions

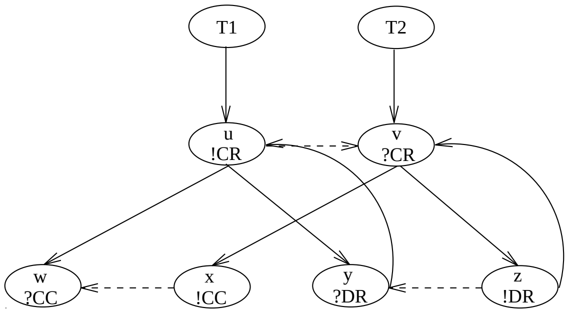

Figure 4. 'Unfolding' an MSC specification into a single ne/sig graph

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

$N_{\mathcal{M}} = (S, C, X, ne, sig, ST, stype, Top, Bottom)$

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 labeling function for the sig edges, and Top and $Bottom$ the labels for the start and end nodes.

5. FROM NE/SIG GRAPHS TO GLOBAL STATE TRANSITION GRAPHS

The ne/sig graph represents an entire MSC specification by a single graph. In order to obtain a finite-state 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[5]. We require that there must be a finite number of global states.

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.

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

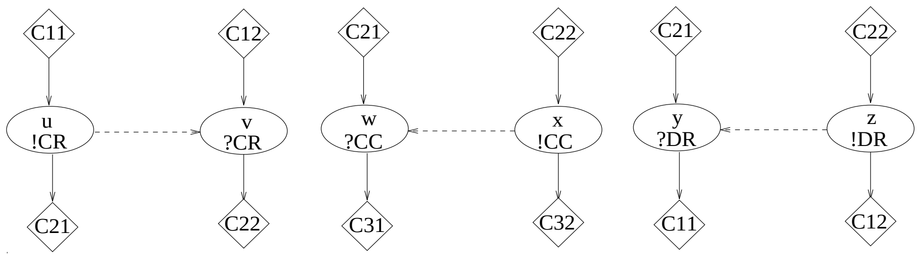

Figure 5. ne/sig graphs with conditions

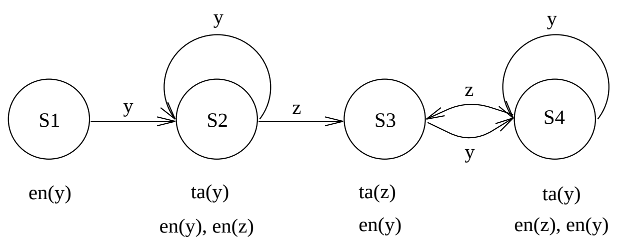

Figure 6. Global State Transition Graph for MSC I

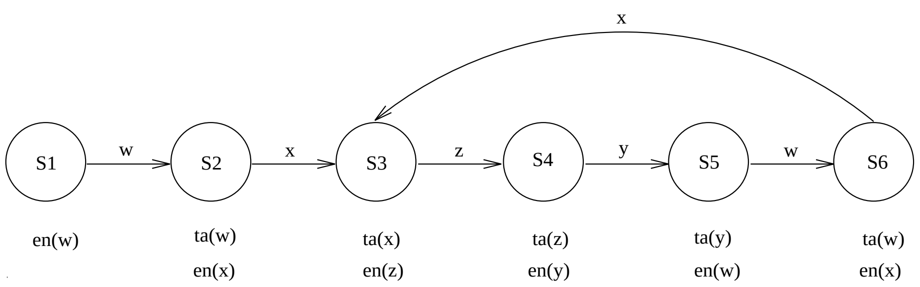

Figure 7. Global State Transition Graph for MSC II

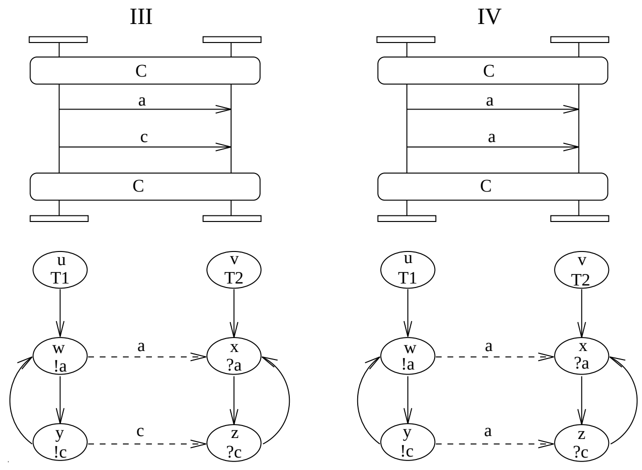

Figure 8. MSCs III and IV and corresponding ne/sig graphs

We shall walk through the derivation of the GSTG for MSC II (Figure 2) given in Figure 7, 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 (the 'program counter'), and sig edges occurring in the state may be thought of as signals sent but not yet received. In state q_0 (labeled $S1$ in Figure 7) 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 7 is annotated with the list of actions enabled (*en()*) and taken(*ta()*) in each state.

Figure 6 shows the GSTG for MSC I. It should be noted that as a result of our finite-state requirement, which inhibits the use of signal queues, no history information on how many messages of one type have been sent is carried along the computation. Consequently, a single `receive` may disable repeated `send`s of one type, as it can be seen in the GSTG for MSC I where node z is not enabled in $S3$. Furthermore, as we argue in [35], MSCs I and IV (see Figure 8) are semantically distinct.

Differences for Synchronous Communication. The definition of transition for synchrony must reflect that sending and receive-ing is a single atomic action, rather than two ordered, separated atomic actions as for asynchrony, and that both participating processes undergo a local state transition at a synchronous event.

To accommodate these considerations, the following modifications must be made. Firstly, a synchronous send and the corresponding receive are *enabled* in a state S if and only if *both* nodes have in-edges in S. This corresponds to the requirement that both sends and receives block until both are ready. Secondly, on a transition through an enabled synchronous event from S to S', ne edges are added and deleted from S as for asynchrony, but for *both* the send and receive events. Furthermore, the sig edge is not added to the current state.

The intuition behind this should be clear. Both send and receive events occur simultaneously in a synchronous communication, therefore a transition occurs through the events in both participating processes. Further, the communication is atomic so no sig edge needs to be added to the current state to denote a send that has not been received.

GSTGs can be Complicated. It should be no surprise that GSTGs can rapidly become very complicated, for example the GSTG for MSC III in Figure 8 has 19 states (see [35]). This is partly due to the asynchronous communication, and partly to interleavings of non-related events. MSC II and MSC III are similar, differing only in that the second message goes in opposite directions. In MSC II this forces a unique execution sequence, and the GSTG is correspondingly simple (Figure 7). However, in MSC III, the two sends might occur before either receive, or alternatively sends and receives 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. We use it later formally to relate liveness properties as expressed in temporal logic or by Büchi automata to MSCs.

6. 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. We now turn to the definition of end-states.

Definition of Global State Automaton. Let M denote a MSC specification and $GSTG_M$ the corresponding global state transition graph. We can define a Büchi automaton which transits between global system states, by adding to $GSTG_M$ 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* $GSTG_M = (Q, q_0, T_M)$ is $A_M \triangleq (Q, q_0, T_M, F)$, where $F \subseteq Q$ is a set of *final states*. Acceptance is Büchi acceptance [45], 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 9 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

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

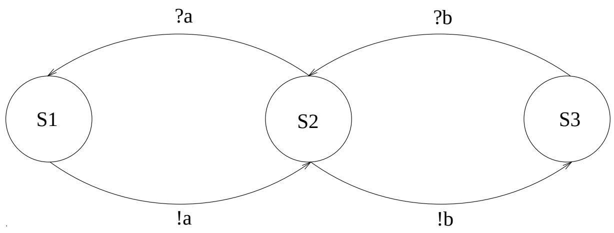

Figure 9. Global state transition graph

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 characteristics assumed for a particular MSC specification. [35] contains examples of liveness properties and their relation to end-state selection.

Liveness and Synchronous Communication. The question of liveness properties changes radically if synchronous sig edges are included. For example, all sends and receives preceding a synchronous event in either process must have occurred before the synchronous event occurs, and there is no question that a synchronous `send` event may not be accompanied by a corresponding `receive`, as there is for asynchrony. It is beyond the scope of this paper to discuss these issues in detail here. We refer the interested reader to the full report [35] for details.

7. MSCS AND THEIR CONNECTION TO TEMPORAL LOGIC

In the last section we noted that liveness properties may have some bearing on the definition of the end-state set of the automaton. A discussion of the use of Büchi automata to specify such properties of distributed systems can be found in [4] and [5]. 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 definition. We remain informal here, referring the reader to [35] for more precision. We base our temporal logic interpretation on the Manna-Pnueli approach [40].

Basic Transition Systems. Following [40] 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.* 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 [40]. An infinite state sequence $\sigma = s_0, s_1, \ldots$ is a *computation* iff s_0 is the initial sate 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 (has been) *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 5. Similarly, a transition is *taken* in a state if that transition leads to the state from an immediately preceding state (notice the 'past tense' sense of the predicate *taken*). Details may be found in [35].

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. [39]. The language has state predicates *en(τ)* and *ta(τ)* as only basic propositions, includes the Boolean connectives (we use just $\neg$ and $\vee$ for simplicity), and the temporal operators $\Diamond$ (eventually), $\Box$ (henceforth), $\diamondminus$ (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 $(\sigma, i) \models p$, i.e. that formula p is *satisfied* in position i of sequence σ.

8. LOGICAL PROPERTIES OF MSC SPECIFICATIONS

We can now give examples of liveness properties expressed in temporal logic which can characterize MSC specifications. The classification of properties as *recurrence* and *reactivity* refers to the classification in [39].

Some Potential Liveness 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 this 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.

9. CONCLUSIONS

We have presented a semantics for Message Sequence Charts that relies upon interpreting an MSC specification as a single ne/sig graph, and then interpreting this graph as a global state transition system. This is almost a Büchi automaton, and we complete the definition of an automaton by considering the end-state definition, which we pointed out depends on liveness properties. Such properties are not usually given explicitly in an MSC specification, and by means of an interpretation of the global state transition graph as a model for temporal logic, we showed how liveness information expressed by temporal logic formulas can be used to enhance MSC specifications. We argued that given our requirements for a semantics, that it be an automaton interpretation, and that it be finite-state, the Büchi automaton model is a natural interpretation.

Acknowledgements
We thank Dr. Ekkart Rudolph, Professor Ken Turner, Philippe Oechslin and the referees for their helpful commentary on this work. Thanks also to Jens Grabowski and Robert Nahm, for discussions when we started this work under contract 233 of the Swiss PTT to the University of Berne, Project Head Professor Dieter Hogrefe.

REFERENCES

1. Siemens AG. *EWSD Softwareentwicklungshandbuch (Software Development Handbook), Kapitel B, Register 6, SDL Diagramme.* Siemens AG, München (Munich), 1988.
2. B. Algayres. personal communication, 1993.
3. B. Algayres, Y. Lejeune, F. Hugonnet, and F. Hantz. The AVALON Project: A VALidatioON Environment for SDL/MSC Descriptions. Unpublished Manuscript. Verilog, Toulouse, France, Feb 1993.
4. B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.
5. B. Alpern and F. B. Schneider. Verifying temporal properties without temporal logic. *ACM Transactions on Programming Languages*, 11(1):147–167, 1989.
6. A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. Research Report 581, IRISA, Rennes, France, 1991.
7. A. Benveniste and P. Le Guernic. Hybrid dynamical systems theory and the SIGNAL language. *IEEE Transactions on Automatic Control*, 35(5):535–546, May 1990.
8. A. Benveniste, P. Le Guernic, and C. Jacquemot. Synchronous programming with events and relations: the SIGNAL language and its semantics. *Science of Computer Programming*, 16(2):103–149, Sep 1991.
9. G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. Rapport de Recherche 842, Institut National de Recherche en Informatique et en Automatique (INRIA), 1988.

10. G. Berry and G. Gonthier. The Esterel synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.
11. M. Broy. Towards a formal foundation of the specification and description language SDL. *Formal Aspects of Computing*, 3:21–57, 1991.
12. CCITT. Syntax and semantics of synchronous interworkings, formal semantics. Contribution D.96-X/3 to CCITT SG X WP 3 meeting in Geneva, 16-20 September 1992.
13. CCITT. Recommendation Q.65: Stage 2 of the method for the characterization of services supported by ISDN. CCITT, Geneva, 1988.
14. 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.
15. CCITT. Recommendation Z.100: CCITT Specification and Description Language (SDL). CCITT, Geneva, 1992.
16. CCITT. Recommendation Z.120: Message Sequence Chart (MSC). CCITT, Geneva, 1992.
17. 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.
18. 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.
19. A. A. R. Cockburn and W. Citrin. An executable specification language for history-sensitive systems. Technical Report IBM RZ 2162, IBM Rüschlikon Research Laboratory, Zürich, 1991.
20. 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.
21. 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.
22. J.-P. Courtiat. ESTELLE*: a powerful dialect of ESTELLE for OSI protocol description. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing, and Verification VIII*. North-Holland, 1988.
23. J.-P. Courtiat. Estelle and Petri nets: introducing a rendezvous mechnism in Estelle: Estelle*. In M. Diaz, J.-P. Ansart, J.-P. Courtiat, P. Azéma, and V. Chari, editors, *The Formal Description Technique Estelle*, pages 175–203. North-Holland, 1989.
24. 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.
25. J. Grabowski, D. Hogrefe, and R. Nahm. A method for the generation of test cases based on SDL and MSCs. Technical Report IAM 93-010, Institute for Informatics, University of Berne, Switzerland, April 1993.
26. P. Graubmann, E. Rudolph, and J. Grabowski. Towards a Petri Net based semantics definition for Message Sequence Charts. In *Proceedings of the 6th SDL Forum*. North-Holland, 1993.
27. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

28. D. Hogrefe. SDL and OSI: On the use of CCITT-SDL in the context of OSI. Habilitation Thesis, University of Hamburg, 1989.
29. G. J. Holzman. *Design and Validation of Computer Protocols.* Prentice-Hall International, 1991.
30. Inmos Ltd. *The Occam Programming Manual.* Prentice-Hall International, 1984.
31. ISO. Information processing systems - Open systems interconnection - Basic reference model, IS 7498. International Standards Organisation, 1984.
32. ISO. Estelle: A formal description technique based on an extended state transition model, IS 9074. International Standards Organisation, 1987.
33. ISO. Information processing systems - Open systems interconnection - LOTOS : A formal description technique based on the temporal ordering of observational behavior, IS 8807. International Standards Organisation, 1988.
34. ISO. Information processing systems - Open systems interconnection - Service conventions, ISO TR 8509. International Standards Organisation, 1989.
35. P. B. Ladkin and S. Leue. Interpreting message sequence charts. Technical Report TR 101, Department of Computing Science, University of Stirling, 1993.
36. 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.
37. L. Lamport. The temporal logic of actions. Technical Report 79, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, Dec 1991.
38. 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.
39. 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.
40. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.
41. S. Mauw, M. van Wijk, and T. Winter. Syntax and semantics of synchronous interworkings. Technical Report RWB-508-RE-92436, Philips Research, Information and Software Technology, Eindhoven, The Netherlands, Oct 1992.
42. R. Milner. *Communication and Concurrency.* Prentice Hall International, 1989.
43. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall International, 1991.
44. A. S. Tanenbaum. *Computer Networks.* Prentice-Hall International, 2nd edition, 1989.
45. 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.
46. 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.
47. M. Van Sinderen, L. Ferreira Pires, and C. A. Vissers. Automata theoretic techniques for modal logics of programs. *The Computer Journal*, 35(5):478–491, 1992.