An Analysis of Message Sequence Charts

Peter B. Ladkin Stefan Leue

University of Berne
Institute for Informatics and Applied Mathematics
Länggassstrasse 51
CH-3012 Bern, Switzerland
ladkin@iam.unibe.ch, leue@iam.unibe.ch

IAM 92-013

June 1992

Abstract

Message (or Time) Sequence Charts (MSCs) are used in telecommunications system specification. To investigate the meaning of an MSC specification, we found the need to connect MSC specifications with more precise methods such as temporal logic and Büchi automata. Based on an interpretation of a collection of MSCs as a global state automaton, we provide an explicit semantic connection to temporal logic, enabling properties expressed by temporal logic to be added to MSC specifications. Finally, we determine the expressiveness of MSCs by simulating an arbitrary Büchi automaton by an MSC specification.

CR Categories and Subject Descriptors: B.4.4 Formal Models [Input/Output and Data Communications]; C.0 Systems specification methodology; C.2.2 Protocol Verification; D.2.1 Requirements/Specifications [Software Engineering]; D.2.2 Methodologies [Software Engineering]; F.1.1 Automata; F.3.1 Specifying and Verifying and Reasoning about Programs [Logics and Meanings of Programs]; F.4.1 Mathematical Logic
General Terms: Design, Verification, Languages
Additional Key Words: Message Sequence Charts, Temporal Logic, Safety and Liveness Properties

Chapter 1

Introduction

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]. In this paper we attempt to do that for one industrial specification method, Message Sequence Charts (MSCs, also called Time Sequence Charts), used in telecommunications protocol specification.

Amongst general system specifications, telecommunications protocol specifications distinguish themselves by an emphasis on communication between processes rather than on the computation within a process, and also by the relatively simple nature of the messages exchanged. These are sometimes PDUs, or *Protocol Data Units*, sometimes ASPs, or *Abstract Service Primitives.* We shall simply call everything a *signal.* MSCs focus entirely on the signals exchanged, and are regarded as a particularly simple, intuitive way of specifying telecommunications protocols. MSCs are the subject of an international standardisation effort by CCITT [X92]. For use of MSCs in industrial practice, see [AG88], [CCHK91], [CCI88a], and [CCI88b].

We shall argue that this simplicity is obtained at the expense of precision, and suggest a remedy, namely that one can add temporal logic statements to an MCS specification via the connection given in this paper. We shall also show that MSC specifications are much more expressive than they appear, by simulating an arbitrary Büchi automaton with an MSC specification.

[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.

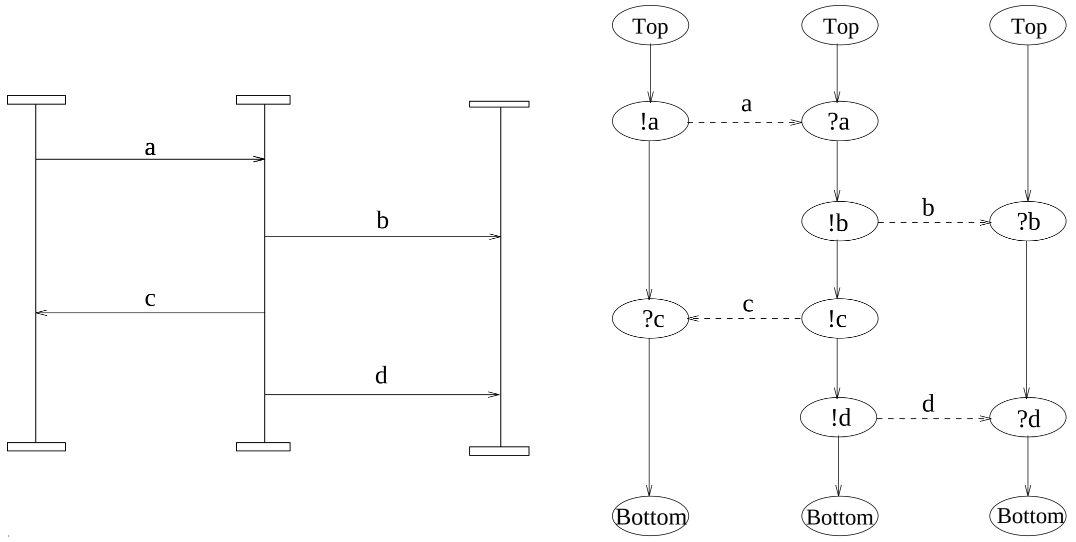

Figure 1.1: A simple MSC and its corresponding ne/sig graph

An MSC Example. Figure 1.1 shows a simple MSC (without conditions), and its syntactic interpretation as a so-called *ne/sig* graph as in [LL92]. 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 this example 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.

Our Motivation. We were led to the analysis of MSC semantics whilst trying to define conformance tests for telecommunications systems by means of MSCs. We found that we could not tell from the 'standard' assumptions which executions were specified by the MSC specification. For example, we could assume that all `send` events had to happen, but did all `receive` events? Could one process send to another infinitely often, but the second not receive any signal at all? (These are both *liveness* properties.) Was the signal sent the 'same' signal as the one received, or did any signal of the same type suffice? (This looks like a requirement on the environment.) These questions led us to define global states and global state transitions, and thus to interpret an MSC as an ω-automaton of global states (i.e. an automaton that accepts infinite sequences of events). Details are in [LL92].

But we found further questions were not answered by this interpretation, or by the MSC specification style. The Büchi automaton[2] we defined was underdetermined by the MSC specification, and the different automata that could be defined depended on safety and liveness properties that were not explicit in that specification[3].

Safety and liveness properties have been investigated through the use of temporal logic and ω-automata. We saw a need to connect MSC specifications with these methods to exploit the precision in stating system properties that is missing from the pure MSC style.

Synopsis of the Paper. We provide a semantic connection between MSCs and temporal logic, and discuss some safety and liveness properties of MSCs in terms of temporal logic formulations. We also show by a simple argument that an arbitrary Büchi automaton may be simulated by an MSC specification. We draw two conclusions from these observations.

1. MSCs as they stand are an incomplete specification method, and MSC specifications need to be supplemented by explicit safety and liveness conditions[4]. Our connection to temporal logic enables this to be done simply by including temporal logic formulae as part of the MSC specification.

2. Despite their apparent simplicity, MSCs are equally expressive with Büchi automata in a precise sense, and more expressive than temporal logic, when communication matters and computation is ignored.

Background Literature. A thorough exposition of the use of temporal logic in system specification may be found in [MP91]. Some knowledge of this reference would be helpful, since our presentation follows their terminology. Details of the connections between system properties, temporal logic, and automata are in [MP90]. An introduction to Büchi automata may be found in [Tho90], and the use of Büchi automata to specify properties of systems may be found in [AS87], [AS89]. Büchi automata express a more complex set of properties than pure temporal logic [Wol83], [VW86], [VW88].

[2]A Büchi automaton is an automaton that can accept infinite sequences. It is similar to a normal finite-state automaton, except for the acceptance condition. An infinite sequence is accepted if some final state is entered infinitely many times during a computation on this sequence.

[3]In [LL92] there is an example which seems to show that MSCs can also make implicit requirements on the environment, that can not be handled by, for example, making the environment an additional process.

[4]In [AS87] it was shown that any property is the intersection of a safety property and a liveness property, so we are safe in saying that this is all that is needed.

Chapter 2

Interpreting MSCs as Automata

In [LL92], we interpreted an MSC specification, which may contain many MSCs linked by conditions, as a single syntactic structure called a *next-event/signal* (ne/sig) *graph*. We then derived a *global state transition graph* (GSTG) from the ne/sig graph. A GSTG is almost a Büchi automaton, missing only the definition of the set of final states. Examples in [LL92] show how the choice of a set of final states for the automaton depended on choices to be made concerning the properties of the communication in the MSCs, which are not explicit in the MSC specification style itself. We summarise this material in this section, omitting formal details, using examples which also occur later in the paper.

MSCs to 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 of ne) labelled *Bottom*[1].

An ne/sig graph corresponding to a simple MSC is syntactic sugar for the MSC. However, representing entire MSC specifications containing MSCs with conditions as a single ne/sig graph may require branching or looping in an ne/sig graph, which is disallowed in MSCs. For example, Figure 2.1 contains two MSCs with conditions, represented by the elongated symbols labelled **C** spanning the process axes. A condition is like a 'joint' for

[1]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.

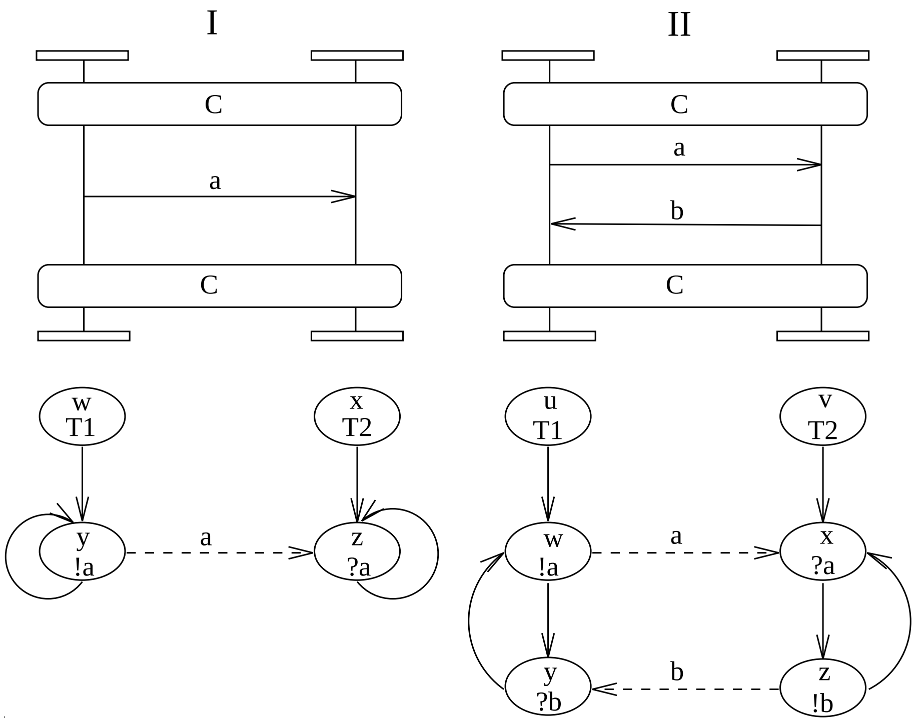

Figure 2.1: MSCs I and II and corresponding ne/sig graphs

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.

Conditions may also be used to specify non-determined behaviour, as in Figure 2.3. At the condition **C2**, the second process may send a **CC** signal to the first, or alternatively a **DR** signal, before looping back to the beginning (condition **C1**). This gives rise to the branching and looping ne/sig graph in Figure 2.4.

The translation is handled in [LL92] first by translating the MSCs with conditions into ne/sig graphs with conditions (Figure 2.5), 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. It's straightforward, but requires care in the details.

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

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

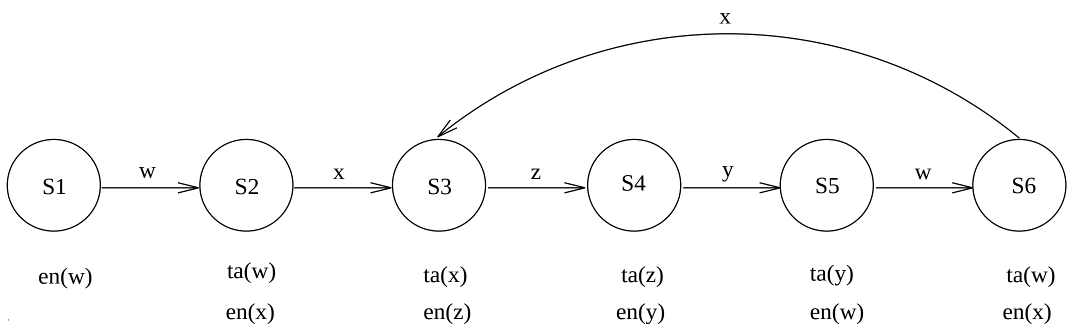

Figure 2.2: Global State Transition Graph for Example II

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.

Ne/sig graphs to GTSGs

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 (GTSG), and is uniquely determined by the MSC specification[2].

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 2.1) given in Figure 2.2, 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 2.2) 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

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

(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 2.2 is annotated with the list of actions enabled and taken in each state. Figure 2.6 shows the GSTG for Example I.

GTSGs can be Complicated. It should be no surprise that GTSGs can rapidly become very complicated, for example the GTSG for Example III in Figure 2.7 is given in Figure 2.8. This is due to the asynchronous communication. For example, Example II and Example III are similar, differing only in that the second message goes in opposite directions. However, in Example II this forces a unique execution sequence, and the GTSG is correspondingly simple (Figure 2.2). However, in example 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 GTSG for every MSC. We use it formally to relate liveness and safety properties as expressed in temporal logic or by Büchi automata to MSCs.

MSC Specifications can 'Count' Receptions. It may also be instructive to compare Example I, Figure 2.1, with Example IV, Figure 2.7. Both examples express a non-terminating `send` of a signal of type a from the first process to the second. We suppose for this brief discussion that all states are end states, so any path through the GSTG is accepted by the corresponding automaton. In Example I, taking a $?a$ action removes the edge $\langle y, z\rangle$ from the state, and thus disables a further $?a$ action until after a further $!a$ action has put the edge back in the state. Thus there may be no two consecutive `receives`, although there may be many consecutive `sends`. In contrast, in Example IV, both $\langle w, x\rangle$ and $\langle y, z\rangle$ may occur in a state (for example, by first performing two `send` actions corresponding to nodes w and y), and execution of a `receive` action corresponding to node x removes $\langle w, x\rangle$ from the state, but not $\langle y, z\rangle$. Thus, a `receive` corresponding to x may be directly succeeded by a `receive` corresponding to z, but then a further `send` must follow before either `receive` is enabled again. Thus, Example IV allows two consecutive receives, but no more, and any number of consecutive sends. It should now be clear how to write down an MSC which enables n consecutive `receives` of a, but no more, for any fixed n.

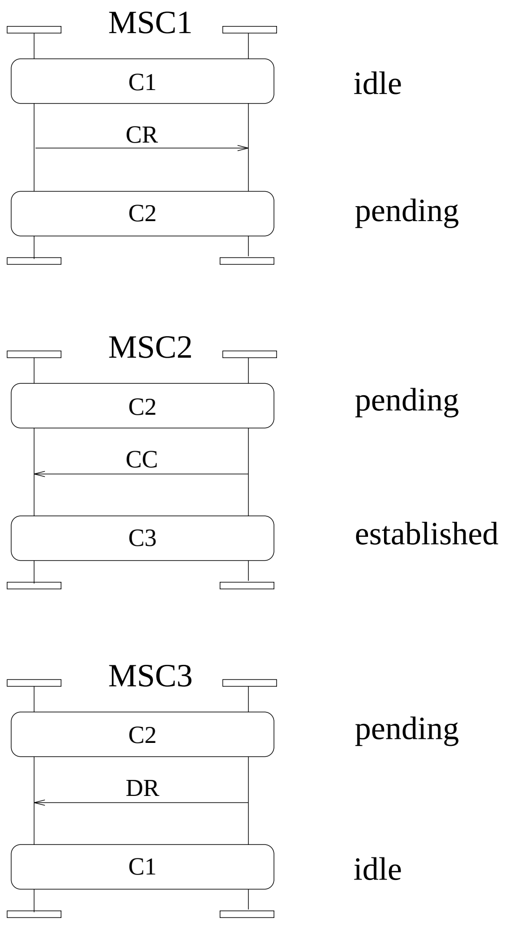

Figure 2.3: MSC specification with conditions

GTSGs Formally. Formally, a GTSG is a tuple $\mathcal{GSTG}_{\mathcal{M}} = (Q, q_0, T_{\mathcal{M}})$ where Q is the set of states obtained from the start state q_0 in the manner described, and $T_{\mathcal{M}}$ is the transition relation, whose elements are triples (G, e, G') where G is a state in Q in which event e is enabled, and G' is the resulting state after e has been taken. Formal details are in [LL92], along with some discussion of end-state selection to achieve various safety and liveness properties.

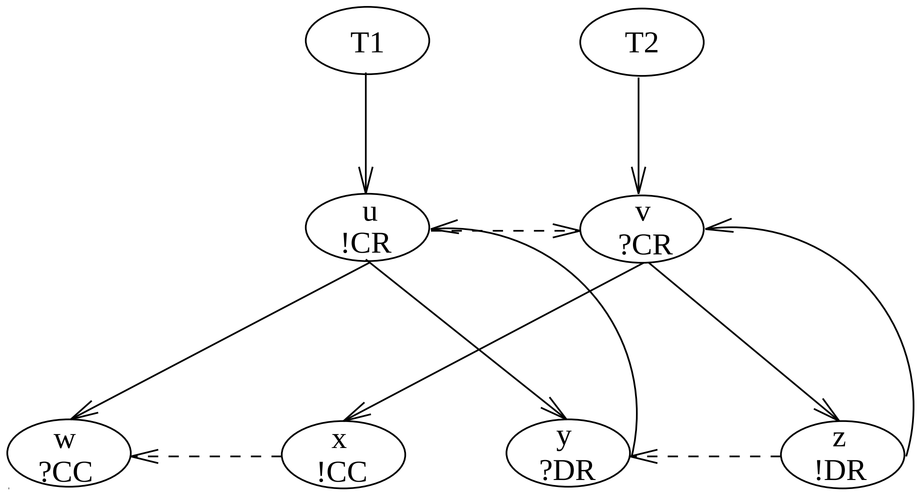

Figure 2.4: 'Unfolding' an MSC specification into a single ne/sig graph

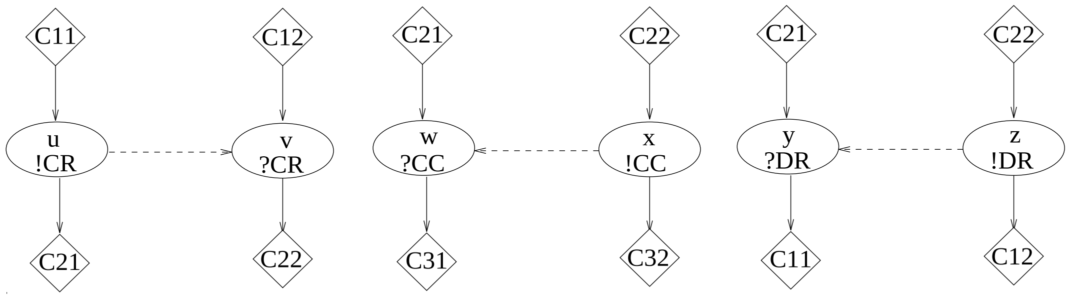

Figure 2.5: ne/sig graphs with conditions

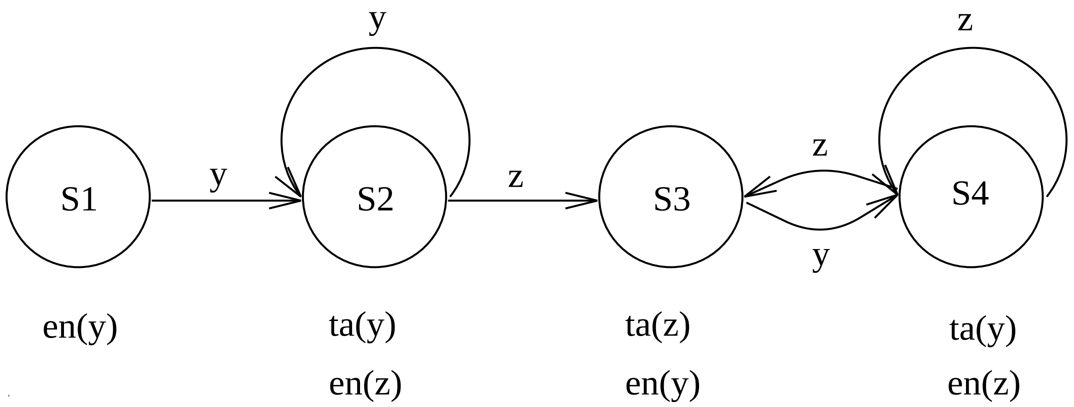

Figure 2.6: Global State Transition Graph for example I

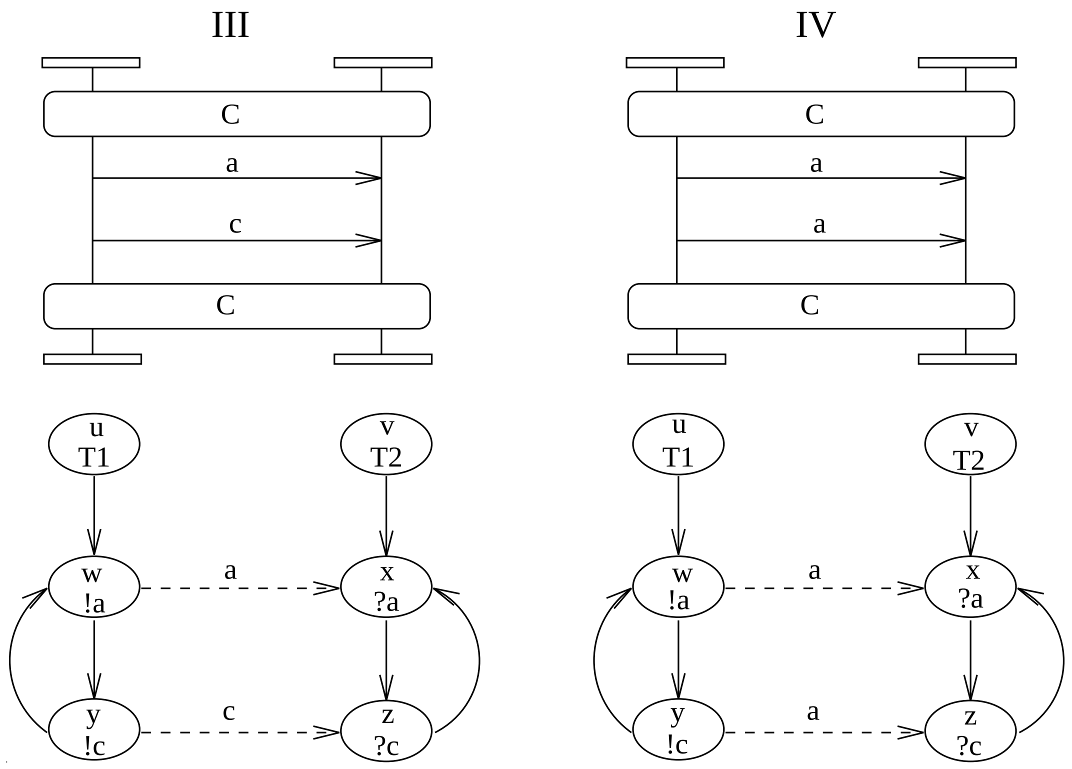

Figure 2.7: MSCs III and IV and corresponding ne/sig graphs

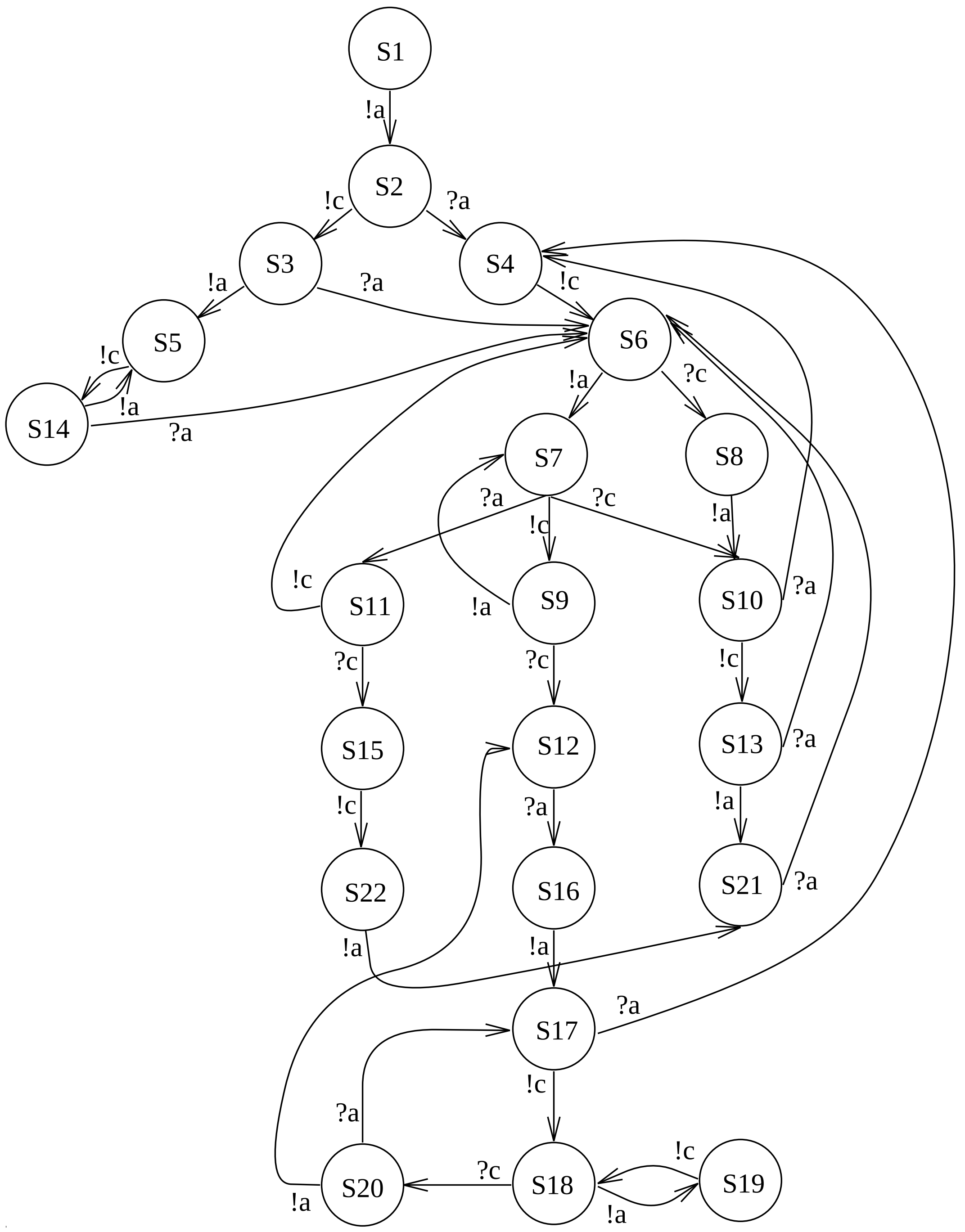

Figure 2.8: Global State Transition Graph for example III

Chapter 3

GTSGs, Transition Systems, and Temporal Logic

Let $N_{\mathcal{M}} = (S, C, X, ne, sig, ST, stype, Top, Bottom)$ denote the ne/sig graph of some MSC specification $\mathcal{M}$ and let $\mathcal{GSTG}_{\mathcal{M}} = (Q, q_0, T_{\mathcal{M}})$ denote the corresponding GSTG. First, we relate the GTSG to a *basic transition system* as defined in [MP91]. Then we consider *state predicates*, *computations* and finally the syntax and semantics of *temporal logic*.

Basic transition system. $\mathcal{BTS}_{\mathcal{M}} \triangleq (\Pi, \Sigma, \mathcal{T}, \Theta)$ is a *basic transition system* corresponding to $\mathcal{GSTG}_{\mathcal{M}}$, where

- Π denotes a set of *state variables*, which is empty in the case of MSC specifications since they do not contain data[1],
- Σ denotes the finite set of states, so $\Sigma = Q$.
- $\mathcal{T}$ denotes a set of *transitions*, $\tau : \Sigma \to 2^{\Sigma}$, for $\tau \in \mathcal{T}$. For $s, s' \in \Sigma$ let $\tau(s) \triangleq \{s' \mid (s, \tau, s') \in T_{\mathcal{M}}\}$ (the symbol τ now has, harmlessly, both a GSTG syntax and a $\mathcal{BTS}$ syntax). For MSCs transitions are events of type $!a$ or $?a$, where a is a signal type.
- Θ denotes an *initial* condition, in our case simply that the initial state *is* the initial state q_0 in $\mathcal{GSTG}_{\mathcal{M}}$.

The states of $\mathcal{BTS}_{\mathcal{M}}$ correspond to global system states of $\mathcal{GSTG}_{\mathcal{M}}$ (they are sets of edges of the *ne* and *sig* relations), and the transitions τ of $\mathcal{BTS}_{\mathcal{M}}$ correspond to communication events of the MSC specification.

[1]The only data in MSC specifications is signal type information, which is encoded in state information.

State Predicates. Manna and Pnueli introduce an assertion which they call the *transition relation*[2] of the form $\rho_\tau : C_\tau(\Pi) \wedge (\bar{y}' = \bar{e})$ describing the change of the values of state variables in state s to their values in state s' into which the system transits from state s by taking transition τ. Since $\Pi = \emptyset$ for MSCs, C_τ is a constant, denoting the *enabling condition* which describes the condition under which the state s may have a successor state by taking the τ transition. $(\bar{y}' = \bar{e})$ stands for a conjunct which expresses the values of a sequence of state variables after the transition has been performed. Since there are no state variables in an MSC specification, this conjunct is vacuous, and so the transition relation ρ_τ is equivalent to the enabling condition C_τ (which is just that there *is* some $\tau' \in \tau(s)$) thus ρ_τ holds in a state s for some transition τ iff there exists $s' \in \Sigma$ such that $s' \in \tau(s)$.

Thus a transition τ is *enabled* in some state s iff $\tau(s) \neq \emptyset$. Conversely, τ is *disabled* in s iff $\tau(s) = \emptyset$. Mapping this to our MSC definitions we have that τ is enabled in state s iff there is some state s' such that $\langle s, \tau, s' \rangle \in T_{\mathcal{M}}$. Consequently, the Manna-Pnueli enabling condition for an action τ is true in precisely those GSTG states in which τ is enabled in the sense of Section 2.

Computations and State Predicates ***en*** **and** ***ta*.** An infinite state sequence $\sigma = s_0, s_1, \ldots$ is a *computation* [MP91], iff

- $s_0 \models \Theta$, which means just $s_0 = q_0$, and
- for all consecutive pairs $s_i, s_{i+1} \in \sigma$ there exists $\tau \in \mathcal{T}$ such that $s_{i+1} \in \tau(s)$.

The indices i of σ are *positions.* Transition τ is *enabled (disabled) at position* i of some computation σ iff it is enabled (disabled) in s_i. We say that transition τ is *taken at position* $i+1$ iff $s_{i+1} \in \tau(s_i)$. We define the predicate *en(τ)* to hold in state $s_i \in \sigma$ iff τ is enabled at position i and we define the predicate *ta(τ)* to hold in state $s_i \in \sigma$ iff τ is taken at position i[3]. As noted, these definitions cohere with our former GSTG definitions. In Figures 2.2 and 2.6, we annotate each state with the instances of *en* and *ta* that are true in that state.

Temporal Logic. We define a temporal logic in the usual way, following [MP90]. The language has state predicates *en(τ)* and *ta(τ)* as 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 $\mathcal{S}$ (since).

[2]We will show that this notion is simplified for MSCs, so there will be no confusion with our notion of transition relation for GTSGs.

[3]In order to avoid notational confusion with our definitions for MSCs we distinguish our notation slightly from the notation used in [MP91].

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.

- If p is a basic assertion, then $(\sigma, i) \models p$ *iff* p is true in s_i as defined above.
- $(\sigma, i) \models \neg p$ *iff not* $(\sigma, i) \models p$
- $(\sigma, i) \models p \vee q$ *iff* $(\sigma, i) \models p$ *or* $(\sigma, i) \models q$
- $(\sigma, i) \models \Diamond p$ *iff* for some $j \geq i$ $(\sigma, i) \models p$
- $(\sigma, i) \models p \mathcal{S} q$ *iff* for some $k, 0 \leq k \leq i, (\sigma, k) \models q$,
  and for every j such that $k < j \leq i$, $(\sigma, j) \models p$
- $(\sigma, i) \models \ominus\, p$ *iff* $i > 0$ *and* $(\sigma, i-1) \models p$

As syntactic abbreviations we introduce the following notation.

- $\Box p \triangleq \neg \Diamond \neg p$
- $\diamondminus p \triangleq \mathit{true}\ \mathcal{S}\ p$

We say that a formula p *holds* on sequence σ iff $(\sigma, 0) \models p$, that it is *satisfiable* iff it holds for *some* computation, and *valid* iff it holds for *all* computations.

Chapter 4

Logical Properties of MSC specifications.

We can now state some examples of properties in temporal logic which can characterise 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. We omit proofs.

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 behaviour alternative (some branch is taken rather than another) in the process's control flow.

   $$\Box(ta(x) \supset \diamondminus en(y))$$

   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.

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

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

Some Potential Requirements on MSC Specifications. Some liveness properties are not automatically fulfilled by an MSC specification $\mathcal{M}$. In [LL92] it was noted that some of these properties were definable by making different selections of the set of final states of a Büchi automaton defined on $\mathcal{GTSG}_{\mathcal{M}}$. Therefore, the MSC specification method may be enhanced by requiring explicit statements of which liveness properties are to be satisfied in a given specification. The most 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, we propose reducing the imprecision in MSC specifications by allowing explicit statements in temporal logic of required properties, for example fairness properties, as part of the MSC specification.

Axiomatising the Logic of MSCs. The question naturally arises, what is the expressive power of MSCs? Since temporal logic has shown its use for defining properties of systems, we could attempt to answer this question by axiomatising the temporal logic formulas satisfied by all MSC specifications. However, we answer this question in what we believe is a more relevant way in the next section, so we do not pursue such an axiomatisation here.

Chapter 5

Abstraction of Automata

Informally, an *abstraction* of a Büchi automaton $\mathcal{A}$ is an automaton $\mathcal{A}'$ whose states are a subset of the states of $\mathcal{A}$, and whose transitions occur not on letters from the alphabet of $\mathcal{A}$, but on sequences of such letters (i.e. words). In other words, the abstraction retains information only on some states, and how one gets from these particular states to others by treating a sequence of transitions of $\mathcal{A}$ as a single transition[1].

An Example. The GSTG for the MSC specification in Figure 2.3 is shown in Figure 5.1. We can form an abstraction of the GSTG, by retaining information only about the global states denoted by conditions C1, C2, and C3, which correspond to the state sets $\{S0, S6\}$, $\{S2\}$, and $\{S4\}$. The abstraction is shown in Figure 5.2[2].

The point of abstractions is that they are in an intuitive sense a summary, a less complex version, of the automaton that they abstract. We show below that an arbitrary Büchi automaton is an abstraction of the automaton of some MSC specification, and therefore that MSC specifications are in this sense equally as complex as Büchi automata, and therefore much more expressive than temporal logic [Wol83], under our semantics.

Abstractions Formally. $\langle \mathcal{A}', I, h \rangle$ is an *abstraction* of $\mathcal{A}$ iff

- I is a mapping of the state set of $\mathcal{A}'$ into the set of state sets of $\mathcal{A}$, i.e. I picks out a subset of the states of $\mathcal{A}$ which correspond with a particular state of $\mathcal{A}'$;
- h maps the alphabet of $\mathcal{A}'$ into words in the alphabet of $\mathcal{A}$, i.e. transitions among states in $\mathcal{A}'$ are translated into sequences of transitions of $\mathcal{A}$;

[1]Abstraction is related to notions in Statecharts [Har87], which can in some sense be regarded as a hierarchy of abstractions. However, we don't need all the details of Statecharts for our purposes

[2]This particular abstraction was called the *composition graph* in [LL92], and is used in the formal definition of the 'unfolding' of a set of ne/sig graphs with conditions into a single ne/sig graph.

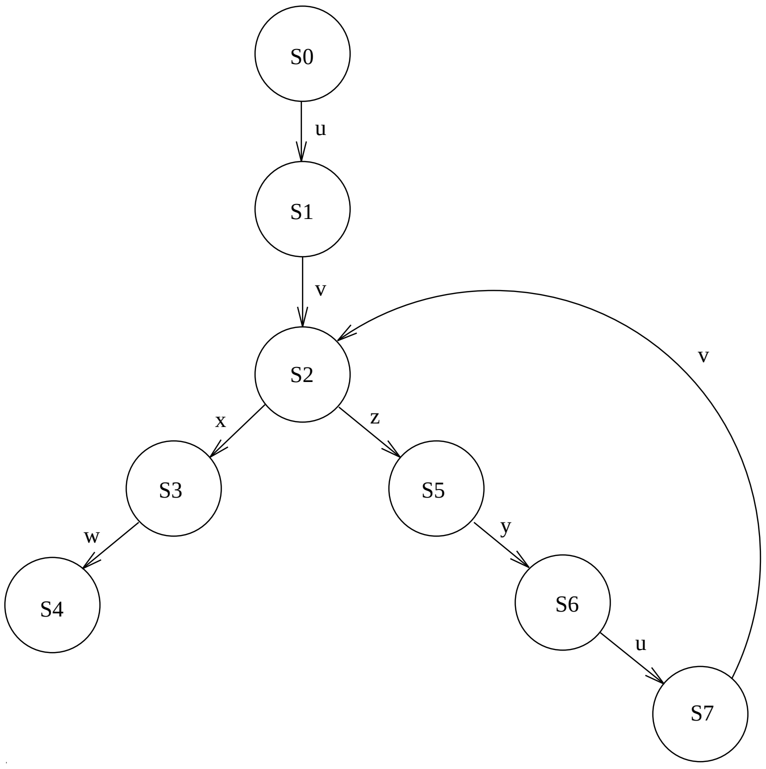

Figure 5.1: Global State Transition Graph

- there is a transition from s to s' on p in $\mathcal{A}'$ iff $h(p)$ is a path from some state in $I(s)$ to some state in $I(s')$ in $\mathcal{A}$,

It is well-known that such an h can be extended into a homomorphism of the words of $\mathcal{A}'$ into the words of $\mathcal{A}$. We shall abuse terminology and refer to $\mathcal{A}'$ alone as an abstraction of $\mathcal{A}$.

A particularly important kind of abstraction is the *one-to-one abstraction*, in which every $I(s)$ is required to be a singleton, i.e. there is only one state of $\mathcal{A}$ corresponding to each state in $\mathcal{A}'$. The abstraction in Figure 5.2 is not a one-to-one abstraction. We categorise the expressiveness of MSCs by the following theorem.

Theorem 5.0.1 *Every Büchi automaton is a one-to-one abstraction of an automaton derived from an MSC specification involving just two processes.*

Proof Sketch. Consider the MSC expressed in Example V in Figure 5.3. It is easy to see that there is only one possible sequence of events, as shown in the GTSG. The following lemma may easily be proved by recursion, and by inspection of the transition between states S1 and S5 in the GSTG.

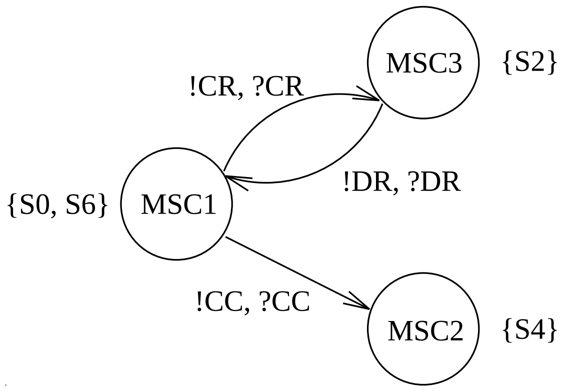

Figure 5.2: An Abstraction Graph

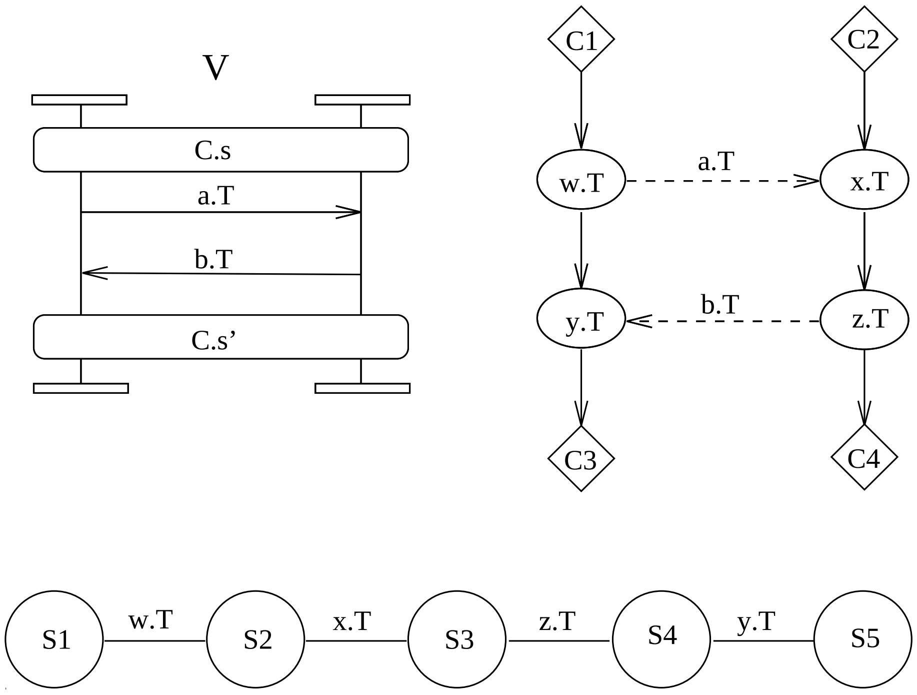

Figure 5.3: Example V and its GSTG

Lemma 5.0.2 *Let $\mathcal{B}$ be a Büchi automaton based on the GSTG of Example V, which additionally satisfies weak liveness. If $\mathcal{B}$ attains the state represented by the beginning condition $C.s$, it will later attain the state represented by the terminal condition $C.s'$.*

To each pair of states s and s' of an arbitrary Büchi automaton $\mathcal{A}'$, and transition T between them, we associate a copy of Example V, with conditions $C.s$ and $C.s'$, and signals $a.T$ and $b.T$. This defines an MSC specification $\mathcal{M}$, with GSTG $\mathcal{GSTG}_{\mathcal{M}}$. We define $I(s) = C.s$ and $h(T) = !a.T?a.T!b.T?b.T$, and finally the global states of the automaton $\mathcal{A}$ derived from $\mathcal{GSTG}_{\mathcal{M}}$ are $\{C.s \mid s \textit{ is a final state of } \mathcal{A}'\}$. Using the lemma, it follows that $\mathcal{A}'$ is an abstraction of $\mathcal{A}$. ▮

A crucial step in this proof is the definition of the endstates of the automaton $\mathcal{A}$ derived from the MSC specification. We are able to do this as we please, only because the end state set is not determined by a pure MSC specification. If some additional requirement on the MSC specification technique is made, for example some fairness requirement for all MSC specifications, then the choice of end state set for the automaton $\mathcal{A}$ will be restricted by this requirement, and we may no longer be able to simulate an arbitrary Büchi automaton. However, we have suggested introducing requirements explicitly in individual MSC specifications as temporal logic statements, so we see no need to make additional implicit requirements on the MSC specification method as a whole. One could further argue that it would be preferable to have as much semantics as possible *explicit* in specifications.

Chapter 6

Concluding remarks

We have concluded in other work [LL92] that MSCs as currently used are an incomplete specification method, requiring additional statements of safety and liveness assumptions. In this paper, we have provided a formal semantic connection between MSCs and temporal logic, enabling MSC specifications simply to be supplemented by explicit safety and liveness conditions expressed in temporal logic. We have recommended enhancing MSC specifications by such temporal logic statements.

We have also shown by a simple argument that an arbitrary Büchi automaton may be simulated by an MSC specification. Hence, despite their apparent simplicity, MSCs are as expressive as Büchi automata in a precise sense (hence more expressive than temporal logic) when only communication acts, and not computation, are considered.

Acknowledgements

Part of this work has been supported by Contract No. 233 of the Swiss PTT to the University of Berne, Project Head Professor D. Hogrefe. We wish to thank the PTT and Professor Hogrefe for their support.

Bibliography

[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, jan 1989.

[CCHK91] A.A.R. Cockburn, W. Citrin, R.F. Hauser, and J. Känel. An environment for interactive design of communication architectures. In *Protocol Specification, Verification and Testing*, volume 11 of *Proceedings of the IFIP WG 6.1 11th Symposium on Protocol Specification, Verification and Testing.* North Holland, 1991.

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

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

[Har87] D. Harel. Statecharts: A visual formalisation for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[LL92] 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. Also submitted for publication.

[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.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, volume 1, Specification. Springer-Verlag, 1991.

[Tho90] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, chapter 4, pages 132–191. Elsevier Science Publisher, 1990.

[VW86] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

[VW88] M. Y. Vardi and P. Wolper. Reasoning about infinite computation paths. Research Report RJ 6209, IBM Almaden Research Center, apr 1988.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

[X92] CCITT SG X. Draft recommendation Z.120: Message sequence chart. Submitted to CCITT, mar 1992.