Characterising Communication Channel Deadlocks in Sequence Diagrams

Bill Mitchell

Department of Computing, University of Surrey

Guildford, Surrey GU2 7XH, UK

w.mitchell@surrey.ac.uk

Draft Version, full version to appear in IEEE Transactions on Software Engineering, VOL. 34, NO. 3, MAY/JUNE 2008, pp 305–320, DOI 10.1109/TSE.2008.28, ref TSE-0046-0207

Abstract

UML sequence diagrams (SDs) are a mainstay of requirements specifications for communication protocols. Mauw and Reniers' algebraic (MRA) semantics formally specifies a behaviour for these SDs that guarantees deadlock free processes.

Practitioners commonly use communication semantics that differ from MRA, which may result in deadlocks. For example FIFO, token ring, etc. We define a process algebra that is an extension of the MRA semantics for regular sequence diagrams. Our algebra can describe several commonly used communication semantics. Regular SDs are constructed from concurrent message flows via iteration, branching, and sequential composition. Their behaviour is defined in terms of a set of partial orders on the events in the SD. Such partial orders are known as causal orders.

We define partial order theoretic properties of a causal order that are particular kinds of race condition. We prove any of the common communication semantics we list either guarantees deadlock free SDs or can result in a deadlock if and only if a causal order of an SD contains one of these types of race condition. This describes a complete classification of deadlocks as specific types of race condition.

1. INTRODUCTION

Scenario based graphical languages, such as message sequence charts (MSCs) [39] and UML sequence diagrams (SDs) [28], are popular for defining requirements specifications. For example, in the automotive industry the dynamic behaviour for the new Media Oriented Systems Transport (MOST) standard has been defined using MSCs [38]. This is a standard agreed between seventeen automotive manufacturers, including BMW, DaimlerChrysler and Jaguar, as well as sixty consumer electronic manufacturers, including Siemens, Philips, and Pioneer.

One reason for the popularity of sequence diagrams is that practitioners find them more intuitive and 'easier' to understand than state machines, [34]. This popularity has lead to the development of verification and test automation tools, such as [6], that can work directly with MSCs and SDs. Such tools then reinforce the use of scenario based specifications.

MSC is the precursor to SD, and was first standardised by the International Telecommunications Union (ITU) in 1992. MSC-96 was given a formal behavioural algebraic semantics by Mauw and Reniers in [23], [24], which we refer to as the MRA semantics. MSC and SD are now mandated by the European Telecommunications Standards Institute (ETSI) for use in the protocol standards making process, [12], [13].

From case studies at Motorola and DaimlerChrsyler [5], we found that practitioners frequently do not use the MRA semantics. It is often the case that they use particular semantics for communication channels between processes and message consumption for input buffers. We found there were a handful of different communication channel semantics that form the majority of these alternative semantics, which will be the focus for this paper. Roughly, these brake down into the following categories. Message passing semantics were almost always one of: asynchronous, synchronous, FIFO and Token-Ring. Whilst most message consumption semantics for input buffers were one of what we termed 'eager' or 'lazy'. For example, the MOST specification uses token ring semantics with 'eager' input buffers rather than the MRA semantics.

The MRA semantics is constructed so that scenario processes do not deadlock. Processes are guaranteed to coordinate correctly according to the specification. However, for the everyday types of semantics we consider here it can well be the case that deadlocks do occur. The fundamental question we address is: what type of behaviour that can now occur as a consequence of such communication channel semantics leads to a sequence diagram deadlock.

Main Results

We first define an operational semantic framework for the various communication semantics that we consider (Section 3), which extends the MRA semantics for partial order scenarios. Such scenarios (defined in Section 2) characterise behavioural semantics as a partial order on the events in the scenario. This partial order is known as the causal order for the scenario. These scenarios allow concurrent threads of activity via parallel constructs, but do not include iteration or branching behaviour.

Once we establish our results for partial order scenarios we extend them to regular sequence diagrams in Section 4. A regular sequence diagram is constructed from a set of partial order scenarios via sequential composition, iteration and branching. For brevity we often refer to a partial order scenario as simply a scenario when this will not cause confusion.

In Section 3 we define a concurrent composition operator $\|_U$ for each of the communication semantics U that we are interested in. Essentially this defines an abstract representation of the various communication semantics that we found were common in the case studies, which were mentioned above.

We define purely partial order theoretic properties of a causal order we call chase and sprint

conditions. These are a refinement of the partial order characterisation of race condition discussed in [25]. In the paper we prove a series of propositions (3.5, 3.6, 3.12, 3.16, 3.18) that characterise what deadlocks are permitted by the various communication channel semantics U. These results prove that a deadlock occurs between partial order scenario processes if and only if the causal order contains either chase or sprint conditions. When this occurs we say the scenario has a chase or sprint condition.

In Definition 4.5 we formally define the notion of a partial order scenario being included in a regular sequence diagram. Intuitively this defines when a scenario describes a specific set of choices for the all the branch points in a sequence diagram up to some particular point. We say a sequence diagram includes a chase or sprint condition if the diagram includes a scenario that has a chase or sprint condition.

Proposition 4.6 proves that a deadlock occurs in a sequence diagram if and only if it includes a partial order scenario that deadlocks. An immediate corollary is that the only cause of a deadlock in a regular sequence diagram is a chase or sprint condition in one of the underpinning causal orders. That is a deadlock occurs in a regular sequence diagram if and only if it includes a chase or sprint condition.

Hence, for the common types of communication semantics that we consider, deadlocks are uniquely determined by partial order theoretic properties of the underpinning causal orders. Further, we can say that different types of race condition in those causal orders completely determine what deadlocks result from communication channel behaviour.

The results reported here grew out of case studies with Motorola and DaimlerChrsyler. They lead to a prototype sequence diagram analysis tool MINT reported in [5], which found errors in approximately one in five sequence diagrams in an early draft version of MOST.

Related Work

[4], [35] contain good surveys of work related to scenario based reasoning. There are many issues relevant to the verification of protocols expressed as UML/MSC diagrams that have been studied. [1], [15], [31], amongst others, have considered verification of logical properties for languages defined by MSCs and MSC-Graphs. [9], [10], [21], [20], [27], [31] consider various different compositional semantics for message sequence charts in order to construct state machines from MSCs and MSC-Graphs. Other work has considered how to interpolate missing

requirements from scenario based specifications [2], [3], [7], [22], [35]. This work is useful both in verifying a system and in synthesising a more complete specification. [36] describes a different approach to synthesis where safety properties are used to determine how scenarios are combined into Modal Transition Systems. [3] is the seminal work that first considered the realisability of collections of MSCs.

Research into automatic test generation from partial order scenarios is an active research area [6], [8], [11], [29]. Amongst others, [30] considers how to reverse engineer a set of scenarios from source code that can then be used for test purposes in an automated test execution environment. [7] has researched error detection in MSCs that are due to concurrent aspects of the scenarios which are caused by a lack of coordination between processes.

The seminal paper to consider race conditions in MSCs was [16]. They characterise the idea of a race condition as a disparity between the causal order on events and an implementation ordering of events. [25], [26] considered issues surrounding ambiguous scenarios. They proved that when resolving race conditions by altering message flows, there exists a unique minimal extension of the original scenario that removes all race conditions.

Live sequence charts (LSCs) [17] are a variation on mainstream MSC/UML scenarios. It is possible to synthesise state machines from LSCs [18], [19], [32], [33], just as with sequence diagrams and MSCs. One of the aims for LSCs has been to allow greater expressitivity. For example, by permitting exemplary and mandatory behaviour to be annotated directly within a scenario. At present LSCs do not have the same following in industry as they have in academia. Also, as mentioned above, MSC/UML SDs are used by a variety of international standards bodies whereas LSCs have not yet gained that level of institutional support.

Graphical Notation

In the paper we will use UML sequence diagrams (SDs) as the graphical language for describing partial order scenarios. We will assume the reader is broadly familiar with the basic concepts of UML SDs. In this section we briefly describe the semantics for those aspects of SDs that we use in the paper.

Consider the SD depicted graphically in Figure 3. Each vertical line describes the time-line for a process, where time increases down the page. Messages are depicted by arrows. Each message

m defines a pair of events $(!m, ?m)$, where $!m$ is the send event for m, and $?m$ is the receive event for m.

The distance between two events on a time-line does not represent any literal measurement of time, only that non-zero time has passed. Events on the same time-line are ordered linearly down the page, except where they occur within a coregion or distinct threads of a parallel construct. Within a coregion events are not locally ordered. Each coregion can only occur on a single time line. It is depicted by a short dashed line delineated by short horizontal lines.

A parallel construct in an SD, denoted by keyword `PAR`, describes a set of interleaving threads that occur in the diagram. Horizontal dotted lines delineate the different threads. Hence, events from one thread are not causally ordered with respect to events from any other thread. Figure 3 contains a parallel constructs split into three threads. The bounding box of a parallel construct has no effect on the ordering of events, it solely delineates the scope of the concurrent threads. Events within a particular thread are ordered in the usual way. Branching in a sequence diagram is represented by the `ALT` construct. Figure 3 contains an `ALT` construct with two possible choices within it. There may be any number of choices within an `ALT` and they are mutually exclusive. Iteration is given by the `loop` construct. This has inline-sequential compositional semantics. A `loop` iterates any finite number of times before terminating. Often A system is described as a set of sequence diagrams. We can always regard such a set as equivalent to a single sequence diagram by using the `ALT` construct to combine all the diagrams in the given set.

The UML notation also allows a message to be split into lost and found events. This allows a message to be sent in one scenario and received in another. The send part of the message is represented by a lost event, and the receive part by a found event. Figure 3 contains two lost messages l_0 and l_1. The OMG semantics for lost and found messages does not make any connection between a lost message and its corresponding found message. We regard a lost message as syntactic sugar for a complete message to a special Null process, and vice-versa for found messages. The Null process has the empty causal ordering. This does not alter message flows with regard to deadlocks and is therefore a harmless convention from our viewpoint.

2. PARTIAL ORDER SCENARIOS

In this section we define the causal order for a partial order scenario and its associated semantics. We use the same message semantics as the MSC 2000 standard [39]. Hence, within this section a partial order scenario defines a set of message exchanges between processes with asynchronous communication channels.

Definition 2.1:

- *A partial order over a set E is a binary relation $<$ such that*

  $<$ is irreflexive, i.e. there is no $x \in E$ where $x < x$

  $<$ is transitive, i.e. if $x < y$ and $y < z$ then $x < z$

  $<$ is asymmetric, i.e. there are no elements $x, y \in E$ such that $x < y$ and $y < x$
- *A total order over the set E is a partial order on E where for any two distinct elements a and b, either $a < b$ or $b < a$.*
- *For $x, y \in E$ when it is not the case that $x < y$ we write $\neg(x < y)$.*
- *Two elements x and y of E are unordered if $\neg(x < y)$ and $\neg(y < x)$.*

  We define a set to be unordered if every pair of distinct elements from that set are unordered.

Let $\mathcal{P}$ be a set of processes. A message m between processes is a pair $(!m, ?m)$ where $!m$ is the send event for m, and $?m$ is the receive event for m. Let E be the set of all send and receive events between all processes.

Definition 2.2: A partial order scenario Sc *on processes $\mathcal{P}$ is*

- *a collection of disjoint sets $E(P) \subseteq E$, for each $P \in \mathcal{P}$*
- *a set of partial orders $<_P$, where $<_P$ is a partial order on $E(P)$ and is referred to as the process order for P*

subject to the constraint that for each send event $!m$ in a set $E(P)$ the corresponding receive event $?m$ occurs in some set $E(Q)$. Note it is possible for $P = Q$.

We treat a partial order as a binary relation that can be represented as the set of pairs that are ordered by the relation. Hence we can take the union of partial orders, which is just the set theoretic union of the sets of pairs given by the relevant order relations. Next, we define the causal ordering that represents the behavioural semantics for a partial order scenario.

Definition 2.3: *The causal ordering $<_{\mathcal{C}}$ on a partial order scenario* Sc *is the transitive closure of the relation given by*

$$\bigcup_{P \in \mathcal{P}} (<_P) \cup \{(!e, ?e) \mid !e \in E(P) \text{ and } ?e \in E(Q) \text{ for some } P, Q \in \mathcal{P}\}$$

The set of pairs $(!e, ?e)$ is used to assert that orderings between processes can only be a consequence of message exchanges. Hence, the causal ordering combines process orderings solely through the causality between send and receive event pairs.

Note, it is possible for there to be two events x and y, both in the same process P, where $x <_{\mathcal{C}} y$ but $\neg(x <_P y)$. Without loss of generality we will assume this is not the case from now on. That is, when $x, y \in E(P)$, we assume $x <_C y$ if and only if $x <_P y$. This is acceptable since the causal semantics will only allow events to be ordered as defined by $x <_C y$. We can therefore modify $<_P$ to include any additional orderings $x <_{\mathcal{C}} y$ where $x <_{\mathcal{C}} y$ but $\neg(x <_P y)$. If we do not adopt this convention the notation becomes irksome without giving us any additional benefits. Hence, if we are given a causal ordering it will be straightforward to extract the process orderings from it. The following definition describes the global system behaviour of a partial order scenario that is meant to occur with respect to the causal order. We will refer to this behaviour as the causal behaviour, or causal semantics depending on the context in which we refer to it.

Definition 2.4: *For a causal ordering $<_{\mathcal{C}}$, a causal system trace is a total order extension of $<_{\mathcal{C}}$. For a process $P \in \mathcal{P}$ with process order $<_P$, a trace of P is a total order extension of $<_P$.*

Thus, the causal order defines which events must be ordered with respect to each other in each system trace, and which events must be independent of each other over the set of all system traces. The causal order does not take into account whether it is possible for processes to act in concert to ensure that the causal order is preserved during execution. As we shall see it is quite possible for execution traces to differ from those specified by the causal order.

2.1. Chase and Sprint Conditions

In this section we define the concept of chase and race condition in a partial order scenario Sc. We also motivate the definition with various examples that illustrate different ways in which

chase and race conditions may cause coordination errors between processes. Chase conditions are a refinement of race conditions, discussed in [16] and [25] amongst others.

Definition 2.5: *Let* Sc *be a partial order scenario with causal ordering* $<_{\mathcal{C}}$*, and events* x*,* $?e \in E$*. Let* $![x] = !h$ *if* x *is either* $!h$ *or* $?h$ *for some* h*. A chase exists between* x *and* $?e$ *when*

$$(x <_{\mathcal{C}} ?e) \text{ and } \neg(![x] <_{\mathcal{C}} !e)$$

A race exists between x *and* $?e$ *when*

$$(x <_{\mathcal{C}} ?e) \text{ and } \neg(x <_{\mathcal{C}} !e)$$

Scenario Sc *is race free if and only if for every pair of events* x*,* $?e$*:*

$$(x <_{\mathcal{C}} ?e) \Rightarrow (x <_{\mathcal{C}} !e)$$

Denote the race property by $r(x, ?e, <_{\mathcal{C}})$ *and the chase property by* $\rho(x, ?e, <_{\mathcal{C}})$*. Notice that* $\rho(x, ?e, <_{\mathcal{C}}) \Rightarrow r(x, ?e, <_{\mathcal{C}})$*, so that chase is a stronger condition than race. We use the term sprint condition to refer to a pair of events* x*,* $?e$ *which form a race condition and not a chase condition.*

This definition has refined the notion of race condition into chase and sprint conditions. Below we will look at some examples of how these occur in case studies.

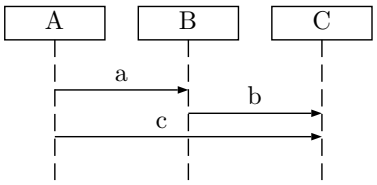

Fig. 1. Simple Example of Chase and Race Condition

In Figure 1 there is a chase between $!b$ and $?c$. There is also a sprint between $?a$ and $?c$, (which is not therefore a chase). This is an interesting example since $?a$ and $!b$ are events on the same life line, with $?a$ preceding $!b$, and yet they cause different race conditions.

Figure 2 shows an example specification taken from a Motorola case study of a telecommunications system used in North America. This has been anonymised to remove all propriety information. Since this scenario specifies system behaviour the causal system traces defined by

this scenario are a subset of the legitimate traces of the system. We will suppose the processes have reached a particular configuration at the start of the scenario (which in the original scenario is described with textual comments), and that the scenario describes how the processes then proceed to reach the next desired configuration at the end of the scenario.

Consider events $?m_i$ and $?m_k$, which are specified by this example to arrive at process E in the order $?m_i <_{\mathcal{C}} ?m_k$. If communication channels between C, D and E are asynchronous, which is perfectly possible for a telecommunications system, it is not possible to ensure $?m_i$ will occur before $?m_k$ in practice because there is no coordination between C, D and E to force this to happen. Hence, latency may cause $?m_i$ to be delayed so that it is received after $?m_k$, even though $!m_i$ is correctly sent before $!m_k$. However, if there is only a single FIFO input channel to E, then we can guarantee that $?m_i$ will occur before $?m_k$ in practice. As a second example consider $!m_k$ and $?m_m$. This is a worse situation, since no matter what latency assumptions we make it will always be possible for G to transmit $!m_m$ too early, so that it arrives before $!m_k$ has occurred. This can occur since there are no messages between D and G which occur after $!m_k$ and before $!m_m$ that could force the necessary coordination to occur.

In Figure 2 we can see race conditions between the following pairs of events

- Sprints: $(?m_c, ?m_i)$, $(?m_i, ?m_k)$, $(?m_o, ?m_q)$, $(?m_t, ?m_v)$
- Chases: $(!m_k, ?m_m)$, $(!m_l, ?m_o)$, $(!m_r, ?m_s)$, $(!m_t, ?m_u)$

This list is not exhaustive, for example $(?m_q, ?m_s)$ is another chase. However, since $?m_q <_{\mathcal{C}} !m_r$ and $(!m_r, ?m_s)$ is already listed it is not useful to include $(?m_q, ?m_s)$ as well. Looking at this list we can see that the sprint conditions can be resolved, for example, by introducing FIFO communication semantics between the appropriate processes. Whereas, the chase conditions will still be present even with, for example, token ring semantics. As we shall prove in later sections, sprint conditions exactly characterise those race conditions that can be resolved by supposing communication channels have something like FIFO semantics, whereas chase conditions can not be resolved in this way. In other words sprint conditions can be resolved by asserting some kind of transmission interdependence between related send and receive messages. Whereas chase conditions can not be resolved in this way.

One way to resolve chase conditions is to allow a process to use lazy message consumption semantics. By this we mean a process has random access to it's input buffer and can delay message consumption from the input buffer until necessary. The structural semantics for lazy

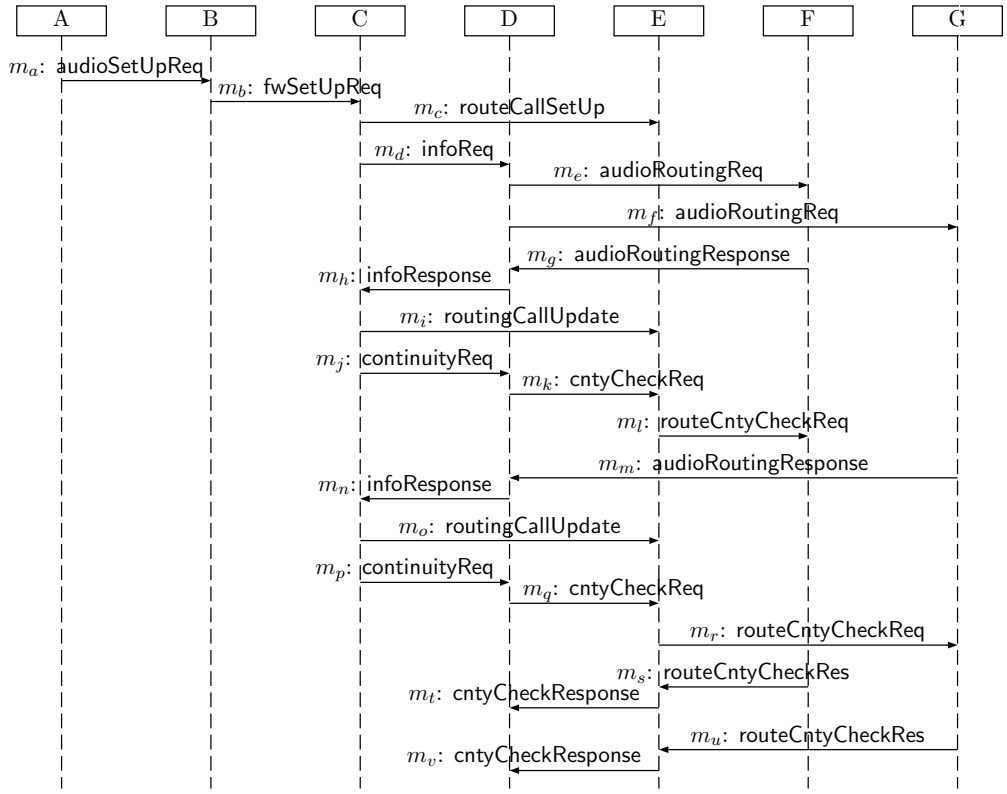

Fig. 2. Example of Multiple Chase and Race Conditions from Motorola Case Study

consumption are formally defined in Section 3. Lazy message consumption generalises the original scenario, in that it results in allowing more system traces than defined by the causal order. Whereas, resolving sprint conditions can be achieved in a way that refines the original system traces.

Figure 3 is a simplified version of an MSC taken from the MOST specification referred to in the introduction. This example has both branching behaviour (shown by the ALT construct, which is short for alternative) and iterative behaviour (shown by the loop construct).

We can consider finite approximations to this scenario that are obtained by unwinding the loop a small finite number, and by looking at different branches that could be taken at each iteration. In doing so we are enumerating the partial order scenarios that are included in Figure 3, (see Definition 4.5). Even before considering the iterative behaviour we can see that there is a sprint between $?m_3$ and $?m_7$. This could be resolved, for example, by adding FIFO semantics to process NetworkSlave_2.

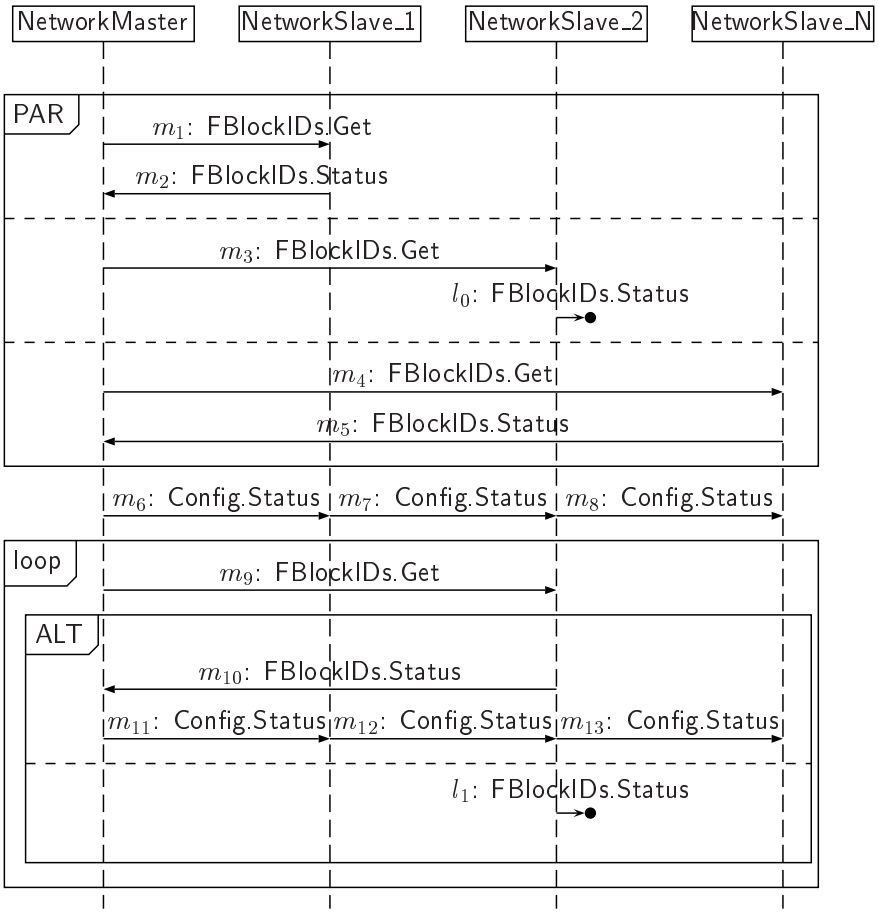

Fig. 3. Simplified SD taken from MOST specification

By adding such semantics we would also resolve the sprint between $?m_3$, $?m_7$ and $?m_9$. Depending on which alternative is taken with each iteration of the loop, there may also be a sprint between consecutive iterations of $?m_9$. This would occur if at some iteration the later branch of the alternative was chosen. Again this would be resolved if NetworkSlave_2 had eager FIFO semantics.

3. General Communication Semantics

The causal semantics in Definition 2.4 describes the global system behaviour of a partial order scenario that is meant to occur, but does not describe a communication semantics between processes that enables them to realise this behaviour. [14] describes such a communication semantics in the form of a process algebra, which extends the MRA semantics. Intuitively we

can summarise the communication semantics from [14] as follows. A process can not send messages directly to another process. Instead, a process can only transmit messages to a global traffic channel, T. Within the process algebra, T is a special process that behaves differently to a normal process. T can always receive messages and stores them in an unbounded random access buffer B, which is represented in the form of a multiset. At the moment a process is specified to receive a message, as defined by the causal behaviour, T removes the relevant message from its buffer and sends it directly to the waiting process. Hence, T acts as a global coordination mechanism that ensures messages always arrive exactly in accordance with the causal ordering. The causal behaviour is equivalent to the globally observed behaviour given by concurrently composing a system's processes and T within the process algebra.

In this section we define structural operational rules that allow us to describe various communication semantics for partial order scenarios. Each type of communication is a modification of the standard causal semantics in [14]. Thus, communication will always consist of processes transmitting messages to a transmission channel T. This channel will then deliver the messages according to the particular semantics being considered.

The causal semantics assumes the traffic channel can act as a global coordination mechanism. The variations defined in this section will not have this property. Hence, it will be possible for processes to become deadlocked if they are not explicitly forced to act in concert to ensure messages arrive in the correct order. The different semantics considered in this section are asynchronous, synchronous, FIFO and token ring communication. We will also consider two variations of FIFO and asynchronous semantics, which we call eager and lazy consumption semantics.

An essential difference from the causal semantics is that processes will now have an input buffer where messages are delivered to. How a message is consumed from the buffer will depend on the particular communication semantics being considered. We will treat message consumption as an internal action that can not be externally observed. We use τ to denote the silent action, which will be generated when a process silently consumes a message from its input buffer. Each operational rule will be controlled by a predicate condition, which is defined in terms of the causal order $<_{\mathcal{C}}$. These will determine exactly how communication occurs. They are designed so that those aspects of the communication semantics we wish to consider can be expressed as properties of the causal order within a partial order theoretic framework.

The structural rules defining the various semantics are given in Figure 4. Each of the constraints InBuf, Trns and Dlv are predicate conditions. By choosing the appropriate values for these conditions we can define the particular communication semantics mentioned above. These choices are given in table 5. The reader will note that the definition of FIFO semantics is a little unusual. We use this format so that we can present all the communication channel semantics in a consistent and concise style. In section 3.3 we will prove that the FIFO semantics here are equivalent to the usual semantics.

Throughout this section we will take Sc to be a partial order scenario on processes $\mathcal{P} = \{P_i \mid 0 \leq i \leq n\}$. For each process $P \in \mathcal{P}$ we define a primitive process term $\mathsf{Pr}(P)$ that describes the behaviour of P. Each primitive term $\mathsf{Pr}(P)$ will be of the form $\mathsf{Pr}(In, S, <_P)$, where In is an input buffer, $S \subseteq E(P)$ is a set of events that are eligible to occur next in a trace, and $<_P$ will define what events will be consecutive to those in S. In is a multiset, as is the buffer B for the transmission channel T.

Definition 3.1: *For a set $S \subseteq E$ and partial order $<$ on E define*

$$\begin{aligned}
\mathsf{n}(S,<) &= \{x \in E \mid \exists\, y \in S :\ y < x, \ \text{ and } \ \neg\exists\, z \in E :\ y < z < x\} \\
\mathsf{m}(S,<) &= \{x \in S \mid \neg\exists\, y \in S :\ y < x\} \\
\mathsf{cns}(a,S,<) &= \mathsf{m}((S - \{a\}) \cup \mathsf{n}(\{a\},<),<)
\end{aligned}$$

The set $\mathsf{m}(S,<)$ contains the minimal elements in S with respect to $<$. The set $\mathsf{n}(S,<)$ is the least upper bound of S with respect to $<$. Notice that $\mathsf{cns}(a,S,<)$ is an unordered set, since the minimal elements of a set are themselves always unordered. If S is an unordered set and $a \in S$, then $S - \{a\} \subseteq \mathsf{cns}(a,S,<)$. In this case $\mathsf{cns}(a,S,<)$ consists of $S - \{a\}$ together with those elements of $\mathsf{n}(\{a\},<)$ that are unordered with respect to $S - \{a\}$.

cns is an abbreviation for consecutive. Suppose we have a causal system trace t that is a total extension of $<$. Let a be some event in t, so that t is of the form $t_0 \cdot a \cdot t_1$ (where $\cdot$ denotes concatenation). Let S be the set of minimal events from the set of all events not in $t_0 \cdot a$. Then t_1 must be of the form $b \cdot t_2$ where $b \in \mathsf{cns}(a,S,<)$, (Lemma 4.2 of [25]). If S contains those events that could occur next at a given point in a system execution and a is the event that then does occur, the set $\mathsf{cns}(a,S,<)$ defines what events may be consecutive to a in a causal system trace.

For the following rules, when $x \in E(P)$ we define $E(x) = E(P)$. We define the concurrent

composition operator $\|$ to be commutative and associative. We use $\texttt{End}(P)$ to denote that process P has successfully terminated.

$$\textsf{Receive} \quad \frac{\phi = \textsf{Pr}(In, S, <_P)}{\phi \xrightarrow{?e} \textsf{Pr}(In \cup \{?e\}, S, <_P)} \; ?e \in E(P) \text{ and } \textsf{InBuf}(In)$$

$$\textsf{Consume} \quad \frac{\phi = \textsf{Pr}(In \cup \{?e\}, S \cup \{?e\}, <_P)}{\phi \xrightarrow{\tau} \textsf{Pr}(In, \textsf{cns}(?e, S, <_P), <_P)} \; \textsf{InBuf}(In)$$

$$\textsf{Send} \quad \frac{\phi = \textsf{Pr}(In, S \cup \{!e\}, <_P)}{\phi \xrightarrow{!e} \textsf{Pr}(In, \textsf{cns}(!e, S, <_P), <_P)} \; !e \notin S \text{ and } \textsf{InBuf}(In)$$

$$\textsf{Transmit} \quad \frac{\phi \xrightarrow{!e} \phi'}{T(B) \parallel \phi \xrightarrow{!e} T(B \cup \{?e\}) \parallel \phi'} \; \textsf{Trns}(B, ?e)$$

$$\textsf{Deliver} \quad \frac{\phi \xrightarrow{?e} \phi'}{T(B \cup \{?e\}) \parallel \phi \xrightarrow{?e} T(B) \parallel \phi'} \; \textsf{Dlv}(B \cap E(?e), !e)$$

$$\textsf{Terminate} \quad \textsf{Pr}(\{\,\}, \{\,\}, <_P) = \texttt{End}(P)$$

Fig. 4. General Communication Semantics for Partial Order Scenario

Notice that the Receive, Consume and Send rules do not involve the transmission channel. They define how a process ordering controls the internal part of message transmission through the input buffer. These rules control process behaviour by ensuring the set of events that are eligible to concurrently occur next is determined by the $\textsf{cns}(e, S, <_P)$ set. This ensures that internally a process behaviour is determined by its process orders $<_P$, which is consistent with causal semantics.

The Transmit and Deliver rules define how the transmission channel then applies a particular communication semantics to messages whilst in transit. These rules are independent of how the process will internally handle sending and receiving messages.

Definition 3.2: *We say that $!e$ is connected to a set of events X, if $!e \in X$ or $?e \in X$. For*

a set $X \subseteq E$*, let* $\mathsf{Sd}(X) = \{!e \mid !e$ is any send event connected to $X\}$*, and let*

$$\downarrow X = \{y \in E \mid \exists\, x \in X : x <_{\mathcal{C}} y\}.$$

Hence, $\downarrow\mathsf{Sd}(X)$ *represents events that are later than any send event connected to* X*.*

	InBuf	Trns	Dlv
Eager Asynchronous (EA)	$In = \{\ \}$	true	true
Lazy Asynchronous (LA)	true	true	true
Eager Fifo (EF)	$In = \{\ \}$	true	$\neg(!e \in\downarrow\mathsf{Sd}(B \cap E(?e)))$
Lazy Fifo (LF)	true	true	$\neg(!e \in\downarrow\mathsf{Sd}(B \cap E(?e)))$
Synchronous (S)	$In = \{\ \}$	$\neg(?e \in\downarrow\mathsf{Sd}(B))$	true
Token Ring (TR)	$In = \{\ \}$	$B = \{\ \}$	$B = \{\ \}$

Fig. 5. Table of Predicates Defining Communication Semantics

When we set condition InBuf to be $In = \{\ \}$ the semantics are defined to be *eager*. In this case a process will deadlock if a message arrives that can not be consumed immediately. Hence, a process must consume messages in an eager manner to avoid deadlock. Note, the deliver rule only permits T to add a message to a process input buffer when it is able to receive a message. As we shall prove below, eager message consumption models the idea that if a message arrives out of order a process will then deadlock. Despite the fact that T will only deliver a message when an input buffer is capable of receiving a message, this does not imply that T acts as a global coordination mechanism. T will deliver a message arbitrarily once it is able, irrespective of whether this is correct with respect to the causal order for a specification. The fact that there is a global delivery system T, does not imply that it must act as a global coordination system.

Definition 3.3: When U *is one of LA, EA, LF, EF, S or TR, then we define* $\|_U$ *to be the concurrent composition defined by Figure 4 with the constraints corresponding to* U *in the table of Figure 5. Let*

$$\mathsf{Pr}(P_i) = \mathsf{Pr}(\{\ \},\, \mathsf{m}(E(P_i), <_{P_i}),\, <_{P_i})$$

Define,

$$\mathsf{Pr}_U(\mathsf{Sc}) = \mathsf{Pr}(P_0) \parallel_U \cdots \parallel_U \mathsf{Pr}(P_n)$$

and $P_U(\mathsf{Sc}) = T(\emptyset) \parallel_U \mathsf{Pr}_U(\mathsf{Sc})$. *Define two sequence of events to be trace equivalent if they are equal once all* τ *actions are deleted from them.*

Define a U *communication trace of* Sc *to be any sequence of events* α *where there is some* α' *trace equivalent to* α *and*

$$P_U(\mathsf{Sc}) \xrightarrow{\alpha'}_\star T(\{\ \}) \parallel_U \mathtt{End}_0 \parallel_U \cdots \parallel_U \mathtt{End}_n$$

For a communication trace α *and* $x, y \in E$, *we write* $x <_\alpha y$ *when* α *is of the form* $\alpha_0 \cdot x \cdot \alpha_1 \cdot y \cdot \alpha_2$.

Examining the communication structural rules in Figure 4 we can see that it is no longer the case that messages are necessarily delivered in the order dictated by $<_\mathcal{C}$. If messages no longer arrive in the right order this may result in deadlock, depending on the particular communication semantics being considered. Inspection of the rules does show that if $x <_C !e$ for any events x and $!e$, then it still is the case that for any communication trace α, $x <_\alpha !e$. This follows since a send event can only be transmitted once all the events before it (with respect to $<_C$) have been consumed. In order to refer to this fact when needed we will formally state it as a proposition.

Proposition 3.4: *For any communication semantics* U, *if* α *is a communication trace of* $P_U(\mathsf{Sc})$ *and there are events* x *and* $!e$ *where* $x <_\mathcal{C} !e$ *then* $x <_\alpha !e$.

Although deadlocks can occur when messages are sent in the wrong order, the lazy semantics has been designed to allow a receiving process the ability to delay the consumption of a message until the appropriate point. Lazy semantics also allows a process to pick any value from its input buffer for consumption. These two facts together mean that processes never deadlock with respect to lazy communication semantics.

Proposition 3.5: *When* U *is any lazy message passing semantics (i.e. when* $\mathsf{InBuf} = \mathsf{true}$*) process* $P_U(\mathsf{Sc})$ *has no deadlocks.*

Proof

Let $\phi_i^k = \mathsf{Pr}(In_i^k,\ S_i^k,\ <_{P_i})$, $\psi_k = T(B_k) \parallel_U \phi_0^k \parallel_U \cdots \parallel_U \phi_n^k$ and suppose there is a deadlock trace $\beta = b_1 \cdots b_k$ where

$$P_U(\mathsf{Sc}) \xrightarrow{\beta}_\star \psi_k$$

Let E_R be the set of receive events in E.

Notice that with any lazy semantics although $T(B_k)$ may have to deliver messages in some constrained way it can always deliver some message as long as its buffer is not empty. Also any process ϕ_i^k can send a message to T so long as there is some send event in S_i^k. Thus a deadlock can occur if and only if $B_k = \emptyset$ and

$$\forall\, 0 \leq i \leq n.\ (S_i^k \subseteq E_R) \quad \text{and} \quad (S_i^k \cap \mathit{In}_i^k = \emptyset)$$

Let

$$?e \in \mathsf{m}(\bigcup_{0 \leq i \leq n} S_i^k, <_{\mathcal{C}})$$

and suppose that $?e \in S_i^k$ for some i, and that $!e \in E(P_j)$ for some j. If $!e$ has not already occurred in β this can only be because there is some $x \in S_j^k$ where $x <_{P_j} !e$. This contradicts that $?e$ is minimal. Hence, $!e = b_r$ for some $1 \leq r \leq k$. Since $B_k = \emptyset$ this can only be true if $?e \in \mathit{In}_i^k$, which is a contradiction. This completes the proof. □

Note the above proposition will also be true if, for example, we consider a lazy version of the Token Ring or Synchronous semantics. The reason we don't consider such lazy alternatives is that Synchronous and Token Ring are meant to work without the need to delay message consumption.

Proposition 3.6: Let U be any eager message passing semantics (i.e. InBuf *is the condition $\mathit{In} = \{\ \}$). If there is a deadlock trace of $P_U(\mathsf{Sc})$ then there are events $?e,\ x \in E$ such that*

$$(x <_{\mathcal{C}} ?e) \quad \text{and} \quad \neg(x <_{\mathcal{C}} !e)$$

That is a deadlock can only occur when $<_{\mathcal{C}}$ contains a race condition.

Proof

Let $\phi_i^k = \mathsf{Pr}(\mathit{In}_i^k,\ S_i^k,\ <_{P_i})$, $\psi_k = T(B_k) \parallel_U \phi_0^k \parallel_U \cdots \parallel_U \phi_n^k$ and suppose there is a deadlock trace $\beta = b_1 \cdots b_k$ where

$$P_U(\mathsf{Sc}) \xrightarrow{\beta}_\star \psi_k$$

First we will prove there must be some $?e \in E$ and some i where $\{?e\} = \mathit{In}_i^k$, and there is some $x \in S_i^k$ where $x <_{\mathcal{C}} ?e$.

As we saw in the proof of proposition 3.5, ψ_k will not deadlock if any process is capable of sending a message. With eager semantics the various ϕ_i^k can always transmit to T as long as their input buffers In_i^k are empty. Consider if a deadlock has been reached because $T(B_k)$ is unable to deliver any of the messages in B_k. From the rules in Figure 4 this can only be if the various ϕ_i^k that are meant to receive one of these events all have a value in their respective input buffers. Thus ψ_k can only be deadlocked if some ϕ_i^k has a non-empty input buffer from which it is unable to consume the contents.
Hence there is some ϕ_i^k where $?e \in B_k$, $?e \notin S_i^k$ and $\{?e\} = In_i^k$. From the definition of the structural communication rules, since $?e$ has not yet been consumed, this implies $?e \in{\downarrow}S_i^k$. Since $?e \notin S_i^k$ that implies there exists some $y \in{\downarrow}S_i^k$ where $y <_{\mathcal{C}} ?e$. Choose the minimum such y, and take this to be the value for x. Note that since $!e$ has already occurred and x has not then $\neg(x <_{\mathcal{C}} !e)$ by proposition 3.4, as required to complete the proof. □

This proposition shows that deadlocks can only occur if $<_{\mathcal{C}}$ does not properly coordinate message passing between processes. Intuitively, it seems quite reasonable that the causal ordering should ensure that when an event is ordered before some receive event it ought also to be ordered before the corresponding send event. Notice the proof of this proposition shows that the eager message passing semantics causes a deadlock if any message is delivered in the wrong order with respect to the causal ordering $<_{\mathcal{C}}$. Thus eager and lazy semantics have opposite policies for handling messages that occur out of order with respect to the causal ordering $<_{\mathcal{C}}$.

Note that from the proof of Proposition 3.6 we immediately have the following corollary.

Corollary 3.7: Suppose that $P_U(\mathsf{Sc})$ deadlocks with trace $\beta = b_1 \cdots b_k$. Then there is

$$P_U(\mathsf{Sc}) \xrightarrow{\beta}_\star \psi_k^U$$

where $\psi_k^U = T(B_k) \parallel_U \phi_0^k \parallel_U \cdots \parallel_U \phi_n^k$, $\phi_i^k = \mathsf{Pr}(In_i^k, S_i^k, <_{P_i})$ and for some i there is $x \in S_i^k$, $\{?e\} = In_i^k$, $\neg(x <_{\mathcal{C}} !e)$ and $x <_{\mathcal{C}} ?e$.

3.1. Eager Asynchronous

With Eager Asynchronous (EA) communication, T has no restrictions on delivering messages, except that the relevant input buffer must be empty. EA semantics restricts the input buffer for

a process to be a single place buffer. The Send rule for EA will not allow a process to pass a message to the transmission channel if there is a message waiting to be consumed. EA semantics assumes scenario processes will spontaneously act in concert to enforce the causal order. Clearly then, so long as processes do not deadlock, the EA semantics will act like the causal semantics. The interesting question is when will processes deadlock.

Proposition 3.8: *The communication traces for* $P_{EA}(\mathsf{Sc})$ *are the causal system traces for* Sc.

Proof

It is clear from the construction of the EA semantics that when $P_{EA}(\mathsf{Sc})$ does not deadlock, there is an equivalence between transitions of $P_{EA}(\mathsf{Sc})$ and $P_c(\mathsf{Sc})$. This follows since each transition

$$T(B) \parallel_c \mathsf{Pr}(S \cup \{!e\}, <_P) \xrightarrow{!e} T(B \cup \{?e\}) \parallel_c \mathsf{Pr}(\mathsf{cns}(!e, S, <_P), <_P)$$

is equivalent to a transition

$$T(B) \parallel_{EA} \mathsf{Pr}(\{\,\}, S \cup \{!e\}, <_P) \xrightarrow{!e} T(B \cup \{?e\}) \parallel_{EA} \mathsf{Pr}(\{\,\}, \mathsf{cns}(!e, S, <_P), <_P)$$

Also, each transition

$$T(B \cup \{?e\}) \parallel_c \mathsf{Pr}(S, <_P) \xrightarrow{?e} T(B) \parallel_c \mathsf{Pr}(\mathsf{cns}(?e, S - \{?e\}, <_P), <_P)$$

is equivalent to the combined transition

$$\begin{aligned} &T(B \cup \{?e\}) \parallel_{EA} \mathsf{Pr}(\{\,\}, S, <_P) \\ &\xrightarrow{\tau \cdot ?e} T(B) \parallel_{EA} \mathsf{Pr}(\{\,\}, \mathsf{cns}(?e, S - \{?e\}, <_P), <_P) \end{aligned}$$

Hence, the communication traces of $P_{EA}(\mathsf{Sc})$ are the same as $P_c(\mathsf{Sc})$. This completes the proof. □

Proposition 3.9: $P_{EA}(\mathsf{Sc})$ *deadlocks if and only if there are events* x *and* $?e$ *such that*

$$(x <_{\mathcal{C}} ?e) \text{ and } \neg(x <_{\mathcal{C}} !e)$$

Proof

Since we already have Proposition 3.6, it only remains to prove the converse to the result. Suppose then that there are x and $?e$ such that

$$(x <_{\mathcal{C}} ?e) \text{ and } \neg(x <_{\mathcal{C}} !e)$$

Suppose that $x \in E(P_i)$. If $?e \notin E(P_i)$ then let $x' \in E(P_i)$ be minimal such that $(x' <_{\mathcal{C}} ?e)$ and $\neg(x' <_{\mathcal{C}} !e)$. Such an x' must exist from the definition of $<_{\mathcal{C}}$. Hence, without loss of generality we may suppose that $?e \in E(P_i)$.

First consider if x is of the form $?g$. Consider those traces generated by allowing $P_{SF}(\mathsf{Sc})$ to execute as follows. We allow processes to execute in a random manner with respect to the EA semantics. However, we restrict T so that if $?g$ is transmitted to its buffer B, then T never delivers $?g$. Effectively this will block any event y where $x <_{\mathcal{C}} y$ from being delivered.

Under these circumstances, either $P_{EA}(\mathsf{Sc})$ will deadlock or we will reach a point where $!e$ is transmitted to T. Suppose that there is a sequence of events β where

$$P_{EA}(\mathsf{Sc}) \xrightarrow{\beta}_{\star} \psi^{EA}$$

with $\psi^{EA} = T(B \cup \{?e\}) \parallel_{EA} \phi_0 \parallel_{EA} \cdots \parallel_{EA} \phi_n$ and $\phi_i = \mathsf{Pr}(In_i, S_i, <_{P_i})$. Either, ϕ_i is deadlocked or has empty input buffer, or can silently consume any message contained in its input buffer. If ϕ_i is deadlocked that completes this part of the proof. Hence, without loss of generality we may suppose $\phi_i = \mathsf{Pr}(\{\,\}, S_i, <_{P_i})$, and that there is some $y \in S_i$ where $?g <_c y$. Otherwise, from the EA semantics, $?g$ would have already occurred, which can not happen because of the restrictions we have placed on T. Hence, we have a transition

$$T(B \cup \{?e\}) \parallel_{EA} \phi_i \xrightarrow{?e} T(B) \parallel_{EA} \mathsf{Pr}(\{?e\}, S_i, <_{P_i})$$

and $\mathsf{Pr}(\{?e\}, S_i, <_{P_i})$ is deadlocked.

Next consider if x is of the form $!g$. We generate traces by allowing $P_{SF}(\mathsf{Sc})$ to execute as follows. We allow all processes except P_i to execute at will, and we place no restrictions on T. However, we do not allow P_i to transmit $!g$. Since $\neg(!g <_{\mathcal{C}} !e)$, this does not prevent $!e$ being transmitted to T at any time. Thus, either $P_{EA}(\mathsf{Sc})$ will deadlock or we will reach a point where $!e$ is transmitted to T. The argument now proceeds just as for the $?g$ case. Thus, we have shown that there will be a deadlock

of $P_{EA}(\mathsf{Sc})$, which completes the proof. □

The Eager Asynchronous semantics illustrates what happens if we try to implement the causal order with the simplest of buffer semantics. When the transmission channel can not enforce the causal ordering then deadlocks will occur exactly when the causal order contains race conditions. Hence, for the EA semantics, race conditions precisely capture when the causal order does not adequately describe coordination between processes in order to avoid deadlock.

3.2. Lazy Asynchronous

We know from Proposition 3.5 that there are no deadlocks for lazy communication semantics. Lazy communication allows messages to be delivered in any order to a process. The process has the responsibility of consuming messages in the correct order with respect to the causal order $<_{\mathcal{C}}$. Since consumption is internal to the process, external observation can only detect that messages are delivered in an arbitrary order. Also, external observation will show that when the correct triggers for some message have arrived (albeit in a random order) then that message will be sent. This turns out to precisely define what communication traces are generated by the Lazy Asynchronous (LA) semantics.

Proposition 3.10: *Let $\alpha = a_0 \cdots a_m$, where $a_i \in E$ for $0 \leq i \leq m$. Then α is an LA communication trace if and only if*

- $\forall\ x, !y \in E,\ x <_\alpha !y \Leftrightarrow x <_{\mathcal{C}} !y$
- $\forall\ !x \in E,\ !x <_\alpha ?x$

Proof

It is clear from the definition of LA semantics, that any LA communication trace must be of the form α as given in the hypothesis.

Suppose then we have a sequence α as in the hypothesis. Let $\alpha_k = a_1 \cdots a_k$. We will prove by induction on k that there are $\phi_i^k = \mathsf{Pr}(In_i^k,\ S_i^k,\ <_{P_i})$, and $\psi_k^{LA} = T(lst_k) \parallel_{LA} \phi_0^k \parallel_{LA} \cdots \parallel_{LA} \phi_n^k$ such that

$$P_{LA}(\mathsf{Sc}) \xrightarrow{\alpha'_k}_\star \psi_k^{LA}$$

for some α'_k trace equivalent to α_k.

The base case is trivial, since the first element of α must be a send event which is minimal with respect to $<_{\mathcal{C}}$. It therefore remains to prove that the above holds for $k+1$.

First consider if a_{k+1} is a receive event $?e \in E(P_i)$ for some i. By definition for α, there is some $r \leq k$ such that $!e = a_r$. Hence, by induction $?e \in B_k$. In which case, since the lazy semantics allows messages to be delivered at any time, we have a transition

$$T(B_k) \parallel_{LA} \phi_i^k \xrightarrow{?e} T(B_k - \{?e\}) \parallel_{LA} \mathsf{Pr}(In_i^k \cup \{?e\},\, S_i^k,\, <_{P_i})$$

Next consider if a_{k+1} is a send event $!e \in E(P_i)$ for some i. By definition of α_k, for any event $y <_{\mathcal{C}} !e$, y must have already occurred in α_k.

If we can not form a transition

$$T(B_k) \parallel_{LA} \phi_i^k \xrightarrow{!e} T(B_k \cup \{?e\}) \parallel_{LA} \mathsf{Pr}(In_i^k,\, \mathsf{cns}(!e, S_i^k, <_{P_i}),\, <_{P_i})$$

Then there is a value $x \in S_i^k$ where $x <_{\mathcal{C}} !e$. Note, from our observation about α_k, there are no send events $!f$ where $x \leq_{\mathcal{C}} !f <_{\mathcal{C}} !e$. Hence x is of the form $?h$, and any value y where $x <_{\mathcal{C}} y <_{\mathcal{C}} !e$ must also be a receive event. We also know that any such y must have already occurred in α_k. From the LA semantics defined by Figure 5 this can only be if every such y is an element of In_i^k. Hence, every such y can be silently consumed by ϕ_i^k. We may therefore replace ϕ_i^k with some term of the form $\mathsf{Pr}(In_i^{k\prime},\, S_i^{k\prime},\, <_{P_i})$, where $?e \in S_i^{k\prime}$. We now do have a transition

$$T(B_k) \parallel_{LA} \phi_i^{k\prime} \xrightarrow{!e} T(B_k \cup \{?e\}) \parallel_{LA} \mathsf{Pr}(In_i^{k\prime},\, \mathsf{cns}(!e, S_i^{k\prime}, <_{P_i}),\, <_{P_i})$$

as required. This completes the induction step, and hence completes the proof. □

3.3. Eager FIFO

The FIFO semantics defined by Figure 5 at first sight seems to have little in common with a more standard definition of FIFO message passing. In this section we will show that from the point of view of deadlock detection they are in fact equivalent. For this section we will abbreviate standard FIFO semantics to SF semantics. Throughout this section let lst be a list of events from E. Let $e :: \mathit{lst}$ be the concatenation of e to the front of lst, and $\mathit{lst}@e$ be the

appending of e to the end of the list. With SF semantics we will use a traffic channel $T(\mathit{lst})$, where lst will now apply the usual FIFO rules to pass messages.

The SF semantics has the Receive, Consume, Send and Terminate rules of Figure 4, which we give the eager semantics. We replace the Transmit and Deliver rules with the following versions:

$$\textsf{Transmit} \qquad \frac{\phi \xrightarrow{!e} \phi'}{T(\mathit{lst}) \parallel \phi \xrightarrow{!e} T(?e :: \mathit{lst}) \parallel \phi'}$$

$$\textsf{Deliver} \qquad \frac{\phi \xrightarrow{?e} \phi'}{T(\mathit{lst}@?e) \parallel \phi \xrightarrow{?e} T(\mathit{lst}) \parallel \phi'}$$

Proposition 3.11: There is a deadlock trace for $P_{SF}(\mathsf{Sc})$ if and only if there is a deadlock trace for $P_{EF}(\mathsf{Sc})$.

Proof

First consider the Dlv constraint for the EF semantics. Unpicking the definition for Dlv we can see that $!e \in{\downarrow}\mathsf{Sd}(B \cap E(?e))$ holds if and only if

$$\forall\, ?f \in B \cap E(?e).\neg(!f <_c !e)$$

Hence, if $?e$ and $?f$ belong to the same process and are both present in T's buffer, and $!f <_c !e$, then $?f$ must be delivered before $?e$. From the definition of SF we can see that if $!f <_c !e$, and both $?e$ and $?f$ are elements of lst, then $?f$ must occur later than $?e$. Hence, $?f$ will be delivered before $?e$. Therefore the SF semantics preserves the EF semantics for delivery.

Let $\phi_i^k = \mathsf{Pr}(\mathit{In}_i^k,\, S_i^k,\, <_{P_i})$, $\psi_k^{SF} = T(\mathit{lst}_k) \parallel_{SF} \phi_0^k \parallel_{SF} \cdots \parallel_{SF} \phi_n^k$ and suppose there is a deadlock trace $\beta = b_1 \cdots b_k$ where

$$P_{SF}(\mathsf{Sc}) \xrightarrow{\beta}_\star \psi_k^{SF}$$

From our earlier remarks we thus have

$$P_{EF}(\mathsf{Sc}) \xrightarrow{\beta}_\star \psi_k^{EF}$$

where $\psi_k^{EF} = T(B_k) \parallel_{EF} \phi_0^k \parallel_{EF} \cdots \parallel_{EF} \phi_n^k$, and B_k is the set of events in lst_k. Hence, by Corollary 3.7, if ψ_k^{SF} deadlocks then there is some ϕ_i^k where $\mathit{In}_i^k = \{?e\}$ and $?e \notin \mathit{In}_i^k$. Hence, ψ_k^{EF} must also be deadlocked.

Consider next if EF deadlocks. So we have a deadlock trace β and process terms ψ_k^{EF}, and ϕ_i^k as above. From proposition 3.6, there are events x, $?e$ where $x <_{\mathcal{C}} ?e$ and $\neg(x <_{\mathcal{C}} !e)$. From Corollary 3.7 we can also suppose that for some i, $In_i^k = \{?e\}$ and $x \in S_i^k$. If $x = ?g$ for some g, then we also know that $\neg(!g <_C !e)$. (Otherwise, EF semantics dictates that $!g$ would have to occur before $!e$. In which case $?g$ would have had to be consumed before $?e$ could be received. Hence, $x \notin S_i^k$, which is a contradiction.) If $x = !g$ for some g, then ϕ_i^k has not yet sent $!g$. Whichever of these cases holds, let $x' = !g$.

Next, we allow $P_{SF}(\mathsf{Sc})$ to execute as follows. Execute any element y where $y \leq_{\mathcal{C}} !e$ whenever possible. Never allow $P_{SF}(\mathsf{Sc})$ to execute x'. Otherwise allow events to be executed at random. Since $\neg(x' <_{\mathcal{C}} !e)$ there will be no reason why we are forced with SF semantics to execute x' in order to ensure some value less than $!e$ can be executed. Therefore, either $P_{SF}(\mathsf{Sc})$ will deadlock, or it must become equal to some process of the form ψ_r^{SF} where $In_i^r = \{?e\}$ and there is some $y <_{\mathcal{C}} x$ where $y \in S_i^r$. In which case, $P_{SF}(\mathsf{Sc})$ is again deadlocked. This completes the proof. □

Given that EF communication semantics deadlock exactly when the SF semantics deadlocks, we next need to characterise exactly when such deadlocks can occur.

Proposition 3.12: $P_{EF}(\mathsf{Sc})$ *will deadlock if and only if there are events* x *and* $?e$ *where*

$$(?x <_{\mathcal{C}} ?e) \text{ and } \neg(![x] <_{\mathcal{C}} !e)$$

Proof

Suppose that $P_{EF}(\mathsf{Sc})$ deadlocks with trace $\beta = b_1 \cdots b_k$. Hence there is

$$P_{EF}(\mathsf{Sc}) \xrightarrow{\beta}_\star \psi_k^{EF}$$

with $\psi_k^{EF} = T(B_k) \parallel_{EF} \phi_0^k \parallel_{EF} \cdots \parallel_{EF} \phi_n^k$ and $\phi_i^k = \mathsf{Pr}(In_i^k, S_i^k, <_{P_i})$. Looking at the proof for proposition 3.11, we must have that for some i there is $x \in S_i^k$, $\{?e\} = In_i^k$, $\neg(![x] <_{\mathcal{C}} !e)$ and $x <_{\mathcal{C}} ?e$. This follows, since in the proof of proposition 3.11, a value x' is constructed that is exactly the value we need for $![x]$. This completes the proof. □

Note, the deadlock condition for proposition 3.12 is a stronger condition than that of proposition 3.6. Thus, we have a complete characterisation of how deadlocks occur for EF communication semantics, and have proved that our representation of FIFO semantics is equivalent, with respect to deadlock detection, to a standard representation. In the following proposition we characterise EF communication traces, which are those traces that describe successful executions of a sequence diagram.

Proposition 3.13: *A sequence $\alpha = a_0 \cdots a_m$ is an EF communication trace if and only if α is a causal system trace, and for all $P \in \mathcal{P}$ and $?x$, $?y \in E(P)$*

$$!x <_{\mathcal{C}} !y \;\Rightarrow ?x <_{\alpha} ?y$$

Proof

First consider where α is an EF communication trace. It is clear from the structural semantics in Figure 4, that any EF communication trace must be a causal system trace. We will prove that $!x <_{\mathcal{C}} !y \;\Rightarrow ?x <_{\alpha} ?y$ by contradiction.

For a contradiction suppose that there are $!x$, $!y \in E$ where $!x <_{\mathcal{C}} !y$ and $\neg(?x <_{\alpha} ?y)$. Since α contains all events in E this implies $?y <_{\alpha} ?x$. The semantics from Figure 4 dictate that $!x <_{\alpha} !y$. Hence it is only possible for $?x <_{\alpha} ?y$ if at some point they are both present in the transmission channel's delivery buffer, and $?y$ is delivered before $?x$. At the point when $?y$ is delivered $?x$ will still be present in the delivery buffer for T. Let $?x, ?y \in E(P_i)$ for some i.

Hence, at some point during the execution of α, $P_{EF}(\mathsf{Sc})$ has transformed into a term of the form $\psi_k^{EF} = T(B_k) \parallel_{EF} \phi_0^k \parallel_{EF} \cdots \parallel_{EF} \phi_n^k$ and $\phi_i^k = \mathsf{Pr}(In_i^k,\, S_i^k,\, <_{P_i})$, where $?x$, $?y \in B_k$. In order for there to be a transition

$$T(B_k) \parallel_{EF} \phi_i^k \xrightarrow{?y} T(B_{k+1}) \parallel_{EF} \phi_i^{k+1}$$

it must be that Dlv holds. Hence, unravelling the definition for Dlv we must have $\neg(!x <_c !y)$. This is a contradiction as required.

Next suppose that α is a sequence as in the hypothesis of the proposition, and we will assume without loss of generality that it does not contain any τ actions. Let $\alpha_k = a_0 \cdots a_k$. Then there is some $\psi(\mathcal{C})_k$ where

$$\psi(\mathcal{C})_k = T(B_k) \parallel_c \mathsf{Pr}(S_0^k, <_{P_0}) \parallel_c \cdots \parallel_c \mathsf{Pr}(S_n^k, <_{P_n})$$

and

$$P_c(\mathsf{Sc}) \xrightarrow{\alpha_k}_\star \psi(\mathcal{C})_k$$

Note, in the equation above $\mathsf{Pr}(S, <)$ represents the behaviour of a process in the causal semantics. At the same time, we use $\mathsf{Pr}(In, S, <)$ to denote the behaviour of a process for the EF semantics.

We can prove by induction on k that if we define

$$\psi_k = T(B_k) \parallel_{EF} \mathsf{Pr}(\{\,\}, S_0^k, <_{P_0}) \parallel_{EF} \cdots \parallel_{EF} \mathsf{Pr}(\{\,\}, S_n^k, <_{P_n})$$

then there is some α'_k equivalent to α_k where

$$P_{EF}(\mathsf{Sc}) \xrightarrow{\alpha'_k}_\star \psi_k$$

The base case is straightforward, so we move on to the induction step. Suppose the above equations hold, we need to show that they also hold for $k+1$. Suppose that $a_{k+1} \in E(P_i)$. Consider first if $a_{k+1} = !e$ for some $!e$. In this case we can trivially prove that the $k+1$ case holds since there is no restriction on EF transmitting messages to T. Without loss of generality we may then suppose that a_{k+1} is of the form $?e$. In which case $?e \in S_i^k$, $S_i^{k+1} = \mathsf{cns}(?e, S_i^k - \{?e\}, <_{P_i})$ and

$$T(B_k) \parallel_c \mathsf{Pr}(S_i^k, <_{P_i}) \xrightarrow{?e} T(B_k - \{?e\}) \parallel_c (S_i^{k+1}, <_{P_i})$$

Hence, for all $?y \in B_k$, $?e <_\alpha ?y$. Therefore by definition of α, for all $?y \in B_k$, $\neg(!y <_{\mathcal{C}} !e)$. Hence from the definition for EF, we have

$$\begin{aligned} T(B_k) \parallel_{EF} \mathsf{Pr}(\{\,\}, S_i^k, <_{P_i}) &\xrightarrow{?e} T(B_k - \{?e\}) \parallel_{EF} (\{?e\}, S_i^k, <_{P_i}) \\ &\xrightarrow{\tau} T(B_k - \{?e\}) \parallel_{EF} (\{\,\}, S_i^{k+1}, <_{P_i}) \end{aligned}$$

This completes the induction step, and hence completes both the proof by induction and the proof of the proposition. □

Proposition 3.13 proves that the EF semantics acts in the usual FIFO manner precisely when the causal order dictates that this must be the case. Proposition 3.11 proves that the EF semantics is equivalent, with respect to deadlock detection, to the usual FIFO semantics. Finally proposition 3.12 gives a purely partial order theoretic characterisation for EF deadlocks.

3.4. Lazy FIFO

Lazy FIFO (LF) semantics asserts that when send events are causally ordered, then their corresponding receive events will be delivered in the same order. The only difference from EF semantics is that input buffers are unbounded and messages can be consumed from them in the order that a process requires. From Proposition 3.5 we know that LF semantics do not deadlock. In the following proposition we describe the LF communication traces.

Proposition 3.14: A sequence of events α is a communication trace for $P_{LF}(\mathsf{Sc})$ if and only if it is a communication trace of $P_{LA}(\mathsf{Sc})$ and for all $P \in \mathcal{P}$ and $?x$, $?y \in E(P)$

$$!x <_{\mathcal{C}} !y \;\Rightarrow ?x <_{\alpha} ?y$$

Proof

Clearly $P_{LA}(\mathsf{Sc})$ can perform any transition that $P_{LF}(\mathsf{Sc})$ can. Hence, any LF communication trace must be an LA communication trace. Proposition 3.13 proved that EF communication traces are exactly the causal system traces that satisfy the partial order constraint of the hypothesis above. This was proved by demonstrating that $P_{EF}(\mathsf{Sc})$ can generate any trace that $P_{\mathcal{C}}(\mathsf{Sc})$ can generate if and only if the above partial order constraint is satisfied. If we replace $P_{EF}(\mathsf{Sc})$ by $P_{LF}(\mathsf{Sc})$, and replace $P_{\mathcal{C}}(\mathsf{Sc})$ by $P_{LA}(\mathsf{Sc})$, then the proof for Proposition 3.13 will go through word for word, which provides a proof for Proposition 3.14. □

This proves that if we generalise EF semantics to allow processes to consume messages when they are required to the result is a FIFO form of LA semantics, as one would expect.

3.5. Synchronous

The intuition for synchronous message passing is that processes wait for an acknowledgement after sending a message before continuing to execute. In MSC/SD this can be explicitly modelled with a suspend region on a life-line, which ends when an acknowledgement is received. Alternately, in SDs there is a graphical notation for depicting a message as synchronous without using a suspend region or explicitly showing an acknowledgement. The intuition here is that a process will not perform any act after sending a message until it is received, and that there

is some observationally silent acknowledgement mechanism that allows the sending process to know when to proceed.

From a trace perspective we can capture this intuition in a partial order theoretic manner that characterises the Synchronous (S) semantics. S semantics dictates that for any message m and S communication trace α, if there is some event e where $!m <_{\mathcal{C}} e$ then $?m <_{\alpha} e$. We prove this formally in Proposition 3.15 below.

Proposition 3.15: A sequence of events α is a communication trace for $P_S(\mathsf{Sc})$ if and only if it is a causal system trace and for all events x and messages m

$$!m <_{\mathcal{C}} x \Rightarrow ?m <_{\alpha} x$$

Proof

$P_{EA}(\mathsf{Sc})$ can execute any transition that $P_S(\mathsf{Sc})$ can. Hence, S communication traces must be EA communication traces. From Proposition 3.8 it follows that S communication traces are therefore causal system traces.

For a contradiction suppose that there are x and m where $!m <_{\mathcal{C}} x$ and $x <_{\alpha} ?m$. Let $\alpha_k = a_0 \cdots a_k$, then for $0 \leq k \leq n$ we can write

$$P_S(\mathsf{Sc}) \xrightarrow{\alpha_k}_{\star} \psi_k^S$$

where $\alpha_k = a_0 \cdots a_k$, $\psi_k^S = T(B_k) \parallel_S \phi_0^k \parallel_S \cdots \parallel_S \phi_n^k$, and $\phi_i^k = \mathsf{Pr}(In_i^k, S_i^k, <_{P_i})$

Suppose that $x = a_k$ for some k. Since $?m$ is not in the sequence α_k and since $!m$ is, we must have $?m \in B_k$. Also if $!m = a_i$, then $?m \in B_j$ for $i \leq j \leq k$.

Consider if x is of the form $?g$. There must have been an earlier transition $\psi_{j-1}^S \xrightarrow{a_j} \psi_j^S$ where $a_j = !g$, and $i \leq j \leq k$. This must have been the result of a transition

$$T(B_{j-1}) \parallel_S \phi_i^{j-1} \xrightarrow{!g} T(B_{j-1} \cup \{?g\}) \parallel_S \phi_i^j$$

From the S structural rules, this transition can only occur when $\neg(?g \in \downarrow \mathsf{Sd}(B_{j-1}))$. If $!m <_{\mathcal{C}} !g$, then by definition $?g \in \downarrow \mathsf{Sd}(B_{j-1}))$. Hence, we must have that $\neg(!m <_{\mathcal{C}} !g)$. This implies that there is a chase condition between $!m$ and $?g$, as we have $!m <_{\mathcal{C}} ?g$ and $\neg(!m <_{\mathcal{C}} !g)$. From Corollary 3.7 this implies α can not be an S communication trace, which is the contradiction we require.

Next suppose that x is of the form $!g$. Then the transition $\psi^S_{k-1} \xrightarrow{a_k} \psi^S_k$ must be due to a transition

$$T(B_{k-1}) \parallel_S \phi_i^{k-1} \xrightarrow{x} T(B_{k-1} \cup \{?g\}) \parallel_S$$

As we saw for the previous case, this can only occur when Trns holds, which can not be true since $!m <_{\mathcal{C}} !g$ implies that $?g \in \downarrow \mathsf{Sd}(B_{k-1}))$. Again we have a contradiction. Hence, any S communication trace must satisfy the partial order constraint.

Next we turn out attention to the converse. Suppose that α is a sequence as in the statement of the proposition. Let $\alpha_k = a_0 \cdots a_k$. We will prove by induction on k that there are $\psi^S_k = T(B_k) \parallel_S \phi^k_0 \parallel_S \cdots \parallel_S \phi^k_n$, and $\phi^k_i = \mathsf{Pr}(\{\,\}, S^k_i, <_{P_i})$ where

$$P_S(\mathsf{Sc}) \xrightarrow{\alpha'_k}_\star \psi^S_k$$

for some α'_k equivalent to α_k. The base case is straightforward, so we move on to the induction step. First consider when a_{k+1} is of the form $!e$. For each $!m <_{\mathcal{C}} !e$, we trivially have $!m <_{\mathcal{C}} ?e$. Hence, each $?m$ is an element of α_k. Therefore, there is no $?m \in B_k$ where $!m <_{\mathcal{C}} !e$. Hence, Trns is true and we have a transition

$$T(B_k) \parallel_S \phi^k_i \xrightarrow{a_k} T(B_{k+1}) \parallel_S \phi^{k+1}_i$$

Next consider when a_{k+1} is of the form $?e \in E(P_i)$ for some i. For S semantics there is no restriction on delivery except that the input buffer be empty, which is the case by the induction hypothesis. Process ϕ^{k+1}_i is of the form

$$\mathsf{Pr}(\{?e\}, S^k_i, <_{P_i})$$

We need to prove that $?e \in S^k_i$. For a contradiction suppose that there is some $x \in S^k_i$ where $x <_{\mathcal{C}} ?e$.

Consider first if x is of the form $!g$. Since $!g <_{\mathcal{C}} ?e$, we have $?g <_\alpha ?e$, by the induction hypothesis. This implies that x must be in α_k which contradicts that it is a value in S^k_i. Consider next if x is of the form $?g$. If $\neg(!g <_{\mathcal{C}} ?e)$ then we have a chase condition which is a contradiction by the proof of Proposition 3.6. Hence, we again have $!g <_{\mathcal{C}} ?e$ and again this leads to a contradiction. Therefore, ϕ^{k+1}_i is able to silently consume $?e$. Thus we can write

$$\phi^{k+1}_i \xrightarrow{\tau} \mathsf{Pr}(\{\,\}, \mathsf{cns}(?e, S^k_i - \{?e\}, <_{P_i}), <_{P_i})$$

This completes the proof by induction, and so completes the proof of the proposition. □

This proposition proves that during any S system execution, that does not deadlock, each process will not perform any action once it has sent a message until the message is received.

Proposition 3.16: $P_S(\mathsf{Sc})$ *will deadlock if and only if there are events* x *and* $?e$ *where*

$$(x <_{\mathcal{C}} ?e) \text{ and } \neg(![x] <_{\mathcal{C}} !e)$$

Note this is exactly the same condition as for Proposition 3.12.

Proof

Suppose that $P_S(\mathsf{Sc})$ deadlocks with trace $\beta = b_1 \cdots b_k$. Hence there is

$$P_S(\mathsf{Sc}) \xrightarrow{\beta}_\star \psi_k^S$$

with $\psi_k^S = T(B_k) \parallel_S \phi_0^k \parallel_S \cdots \parallel_S \phi_n^k$ and $\phi_i^k = \mathsf{Pr}(In_i^k, S_i^k, <_{P_i})$. From Corollary 3.7, we must have that for some i there is $x \in S_i^k$, $\{?e\} = In_i^k$, $\neg(x <_{\mathcal{C}} !e)$ and $x <_{\mathcal{C}} ?e$.
If x is of the form $!g$ then we are done. Suppose then that x is of the form $?g$. Since $\{?e\} = In_i^k$ we know that $!e$ is in α_k. For a contradiction suppose that $!g <_{\mathcal{C}} !e$. This implies that $!g$ is also an element of α_k. Since $!g$ has occured, but $?g$ has not, $?g \in B_k$.
Let $?e = b_j$, for some j. Thus there is a transition

$$T(B_{j-1}) \parallel_S \mathsf{Pr}(\{\,\}, S_i^{j-1}, <_{P_i}) \xrightarrow{!e} T(B_{j-1} \cup \{?e\}) \parallel_S \mathsf{Pr}(\{\,\}, S_i^{k-1}, <_{P_i})$$

It must also be that Trns holds for this transition to occur. That is $\neg(?e \in \downarrow\mathsf{Sd}(B_{j-1}))$. However, $?g \in B_{j-1}$ since $!g$ occurs before $!e$ and $?g$ has not occurred. That implies $?e \in \downarrow\mathsf{Sd}(B_{j-1})$, which is a contradiction as required.
For the converse suppose that $![x] \in E(P_i)$. We allow $P_S(\mathsf{Sc})$ to execute randomly, with the exception that P_i must not transmit $![x]$ to T. Either, $P_S(\mathsf{Sc})$ will deadlock at some point, or eventually $?e$ will be transmitted. At that point we will reach a deadlock as described by Corollary 3.7. This completes the proof. □

We have proved that the deadlock condition for S semantics is exactly the same as for EF semantics. Note however that the traces of these semantics are quite different.

3.6. Token Ring (TR)

TR semantics only allows a single message to be in transit at any time. The concept comes from systems where a virtual token is continually passed around a network ring. When a process holds the token no other process may send a message. Once a message is sent, the process holding the token only releases it once the message is received. The structural communication rules in Figure 4 can simulate this with the constraints given in Figure 5.

The constraints force at most one value to be in the buffer B for the transmission channel T at any time. The Trns constraint ensures that a value can only be transmitted to T when B is empty. The Dlv constraint for TR ensures that a message can not be delivered unless it is the only value in B. Also the constraints force a process to consume messages in an eager fashion. Note, it is possible for a process to send a message to T, which is then delivered to a process Q. It may be that Q will deadlock at this point, but other processes can continue under the TR semantics to send messages. It is also possible that Q does not immediately consume the message, and other processes start to send messages before Q does so. However, consumption is silent and we can suppose without loss of generality that it does occur as soon as possible without affecting the discussion here. Hence, the TR semantics does not completely characterise the intuitive concept of passing a token. However, this only fails when one of the processes deadlocks, and so for the purposes of this paper it adequately characterises token ring semantics.

Proposition 3.17: *A sequence of events $\alpha = a_0 \cdots a_n$ is a communication trace for $P_{TR}(\mathsf{Sc})$ if and only if it is a causal system trace and for all messages m*

$$\forall\, 0 \leq i \leq n.\ (!m = a_i) \Rightarrow (?m = a_{i+1})$$

Proof

By inspection of the constraint on the structural rules for TR we can see that the following holds.

- There can be a transition

  $$T(B) \parallel_{TR} \mathsf{Pr}(In, S, <_P) \xrightarrow{!e} T(B \cup \{?e\}) \parallel_{TR} \mathsf{Pr}(In, \mathsf{cns}(?e, S - \{?e\}, <_P), <_P)$$

  if and only if $B = \emptyset$ and $In = \emptyset$.

- There can be a transition

  $T(B \cup \{?e\}) \parallel_{TR} \mathsf{Pr}(In, S, <_P) \xrightarrow{?e \cdot \tau}_\star T(B) \parallel_{TR} \mathsf{Pr}(In, \mathsf{cns}(?e, S - \{?e\}, <_P), <_P)$

  if and only if $B = \emptyset$ and $In = \emptyset$.

Hence, for any events $!a$ and b there is a transition equivalent to

$$T(B) \parallel_{TR} \mathsf{Pr}(In, S, <_P) \xrightarrow{!a \cdot b}_\star T(B') \parallel_{TR} \mathsf{Pr}(In', S', <_P)$$

if and only if $b = ?a$. This completes the proof. □

Proposition 3.18: $P_{TR}(\mathsf{Sc})$ *will deadlock if and only if there are events* x *and* $?e$ *where*

$$(x <_{\mathcal{C}} ?e) \text{ and } \neg(![x] <_{\mathcal{C}} !e)$$

Note this is exactly the same deadlock condition as for EF and S semantics.

Proof

It is clear from the structural rules for TR, that EF can simulate any transition that TR can. Thus, by Corollary 3.7, if TR deadlocks then so does EF. This proves that if TR does deadlock then the condition above holds.

Suppose then that we are given events x and $?e$ where

$$(x <_{\mathcal{C}} ?e) \text{ and } \neg(![x] <_{\mathcal{C}} !e)$$

First consider if x is of the form $!g$. As we did with the other semantics we can allow $P_{TR}(\mathsf{Sc})$ to execute randomly, but with the restriction that $!g$ is not allowed to be transmitted to T. As with the other semantics this will cause $P_{TR}(\mathsf{Sc})$ to deadlock eventually.

Next consider the case where x is of the form $?g$. From Proposition 3.17 we know there can be no trace where $!g$ occurs after $!e$ and before $?e$. Thus in any trace of $P_{TR}(\mathsf{Sc})$ if $!e <_\beta !g$ then $?e <_\beta !g$, or $P_{TR}(\mathsf{Sc})$ deadlocks before $!g$ can occur. Thus if we have $?g <_{\mathcal{C}} ?e$, but $\neg(!g <_{\mathcal{C}} !e)$ we can allow $P_{TR}(\mathsf{Sc})$ to execute randomly with the restriction that $!g$ is not allowed to be transmitted to T. In which case we must eventually reach a deadlock. This completes the proof. □

4. REGULAR SEQUENCE DIAGRAMS

A regular sequence diagram is constructed from a set of partial order scenarios by combining them with sequential composition, iteration and branching operators. This section extends the earlier operational semantics for partial order scenarios to regular sequence diagrams. The semantics we define are equivalent to those defined in the MRA semantics [23], [24]. We define them in a form that permits us to integrate them with the earlier semantics for partial order scenarios with minimal effort. This section also proves one of the main results of the paper, Proposition 4.6.

Convention: Since we will only be concerned with regular sequence diagrams we will simply refer to them as sequence diagrams from now on.

Definition 4.1: We define a sequence diagram process term as follows. This is defined with respect to the possible communication semantics U given in the table of Figure 5 and using the notation in Definition 3.3.

We assume there is a fixed set of processes $\mathcal{P}$ over which all the sequence diagrams will be defined. If Sd_1 and Sd_1 are sequence diagram process terms then so are

- $\mathsf{Sd}_1 + \mathsf{Sd}_2$ *(alternative operator)*
- $\mathsf{Sd}_1 :: \mathsf{Sd}_2$ *(concatenation operator)*
- Sd_1^{∞} *(loop operator)*

For any partial order scenario Sc with processes $\mathcal{P}$ where $P_U(\mathsf{Sc}) \xrightarrow{\alpha}_{\star} T(B) \parallel_U \mathsf{Pr}'_U$ for some string of events α then Pr'_U is also a sequence diagram process term. As usual we define $+$ to be associative and commutative, and $::$ to be associative.

When α is the empty string we say Pr'_U is an initial term. That is Pr'_U is an initial term when no event has yet occurred in $P_U(\mathsf{Sc})$. Recall from Definition 3.3 that the initial term is denoted $\mathsf{Pr}_U(\mathsf{Sc})$.

A sequence diagram is a process term as above but constructed only from initial terms and the operators $+$, $::$ and ∞.

Intuitively $\mathsf{Sd}_1 + \mathsf{Sd}_2$ is the mutually exclusive choice between alternatives. Graphically this would be shown as an ALT construct. $\mathsf{Sd}_1 :: \mathsf{Sd}_2$ is the inline sequential composition operator. We refer to it as the concatenation operator. This represents the visual idea of concatenating two

sequence diagrams together when they contain the same processes. $\mathsf{Sd}_1 :: \mathsf{Sd}_2$ amounts to the sequential composition of the corresponding processes in the two sequence diagrams. Note, it is quite possible with concatenation for some events within the second sequence diagram to occur before all the events in the first diagram have finished. Sd_1^∞ represents the arbitrary iteration of Sd_1 any finite number of times. Note, we don't need to explicitly define finite iteration since any term formed by finite iteration can be replaced by an equivalent term using sequential composition and branching.

In order to have compact operational semantics for sequence diagrams we define some notation concerning when some or all of the processes in a partial order scenario have ended.

Definition 4.2: *Let* Sc *be a partial order scenario with processes* $\mathcal{P}$*, where* $P_U(\mathsf{Sc}) \xrightarrow{\alpha}_\star T(B) \parallel_U \mathsf{Pr}'_U$ *for some string of events* α*. We refer to* Pr'_U *as a scenario process term.*

If Pr'_U *is of the form* $\mathtt{End}(P) \parallel_U Q$ *for some* Q*, then we say* P *has ended in* Pr'_U*. We say* P *has ended in a sequence diagram process term* Sd *when it has ended for every* Pr'_U *that occurs in* Sd*, which we denote by* $\mathtt{End}(P, \mathsf{Sd})$*. When* P *has ended in* Sd *for every* $P \in \mathcal{P}$ *we say* Sd *has ended. We use* $\mathtt{End}$ *to denote a process term that has ended.*

We formally define the operational semantics for the alternative, concatenation and loop operators in Figure 6. $\mathsf{End\ Alt}$ is the only non intuitive rule in Figure 6. The alternative construct semantics can have subtle consequences. Consider Figure 7, where the first alternative choice contains no actions for process A. If this first alternative is chosen then $!c$ will be the initial event for process A. Moreover, this can validly occur before process C sends event $!b$.

Definition 4.3: *For a sequence diagram* Sd*, string of events* α *and sequence diagram process term* Sd' *we write*

$$T(B) \parallel_U \mathsf{Sd} \xrightarrow{\alpha}_\star T(B') \parallel_U \mathsf{Sd}'$$

when the operational semantic rules in Figures 4 and 6 allow us to transform $T(B) \parallel_U \mathsf{Sd}$ *into* $T(B') \parallel_U \mathsf{Sd}'$ *via the events defined in the string* α*. We define* Sd *to have a deadlock trace when there is a sequence of transitions*

$$T(\{\ \}) \parallel_U \mathsf{Sd} \xrightarrow{\alpha}_\star T(B') \parallel_U \mathsf{Sd}'$$

and there are no (non-τ*) transitions possible for* Sd' *and not all processes have ended in* Sd'*. When* Sd *has a deadlock trace we say* Sd *deadlocks.*

In the following rules a is any event in E.

Alt
$$\frac{T(B) \parallel_U \mathsf{Sd}_1 \xrightarrow{a} T(B') \parallel_U \mathsf{Sd}_1'}{T(B) \parallel_U (\mathsf{Sd}_1 + \mathsf{Sd}_2) \xrightarrow{a} T(B') \parallel_U \mathsf{Sd}_1'}$$

End Alt
$$\frac{\mathtt{End}(P, \mathsf{Sd}_1)}{\mathsf{Sd}_1 + \mathsf{Sd}_2 \xrightarrow{\tau} \mathsf{Sd}_1}$$

Concat
$$\frac{T(B) \parallel_U \mathsf{Sd}_1 \xrightarrow{a} T(B') \parallel_U \mathsf{Sd}_1'}{T(B) \parallel_U (\mathsf{Sd}_1 :: \mathsf{Sd}_2) \xrightarrow{a} T(B') \parallel_U (\mathsf{Sd}_1' :: \mathsf{Sd}_2)}$$

?Concat
$$\frac{T(B) \parallel_U \mathsf{Sd}_2 \xrightarrow{?a} T(B') \parallel_U \mathsf{Sd}_2'}{T(B) \parallel_U (\mathsf{Sd}_1 :: \mathsf{Sd}_2) \xrightarrow{?a} T(B') \parallel_U (\mathsf{Sd}_1 :: \mathsf{Sd}_2')}$$

!Concat
$$\frac{T(B) \parallel_U \mathsf{Sd}_2 \xrightarrow{!a} T(B') \parallel_U \mathsf{Sd}_2'}{T(B) \parallel_U (\mathsf{Sd}_1 :: \mathsf{Sd}_2) \xrightarrow{!a} T(B') \parallel_U (\mathsf{Sd}_1 :: \mathsf{Sd}_2')} \mathtt{End}(P, \mathsf{Sd}_1),\ !a \in E(P)$$

Null Concat
$$\frac{\mathtt{End} :: \mathsf{Sd}}{\mathsf{Sd}}$$

Loop
$$\frac{\mathsf{Sd}^\infty}{\mathtt{End} + \mathsf{Sd} :: \mathsf{Sd}^\infty}$$

Fig. 6. Communication Semantics for Regular Sequence Diagrams

Before we go on to the main result we first show that if we concatenate two partial order scenarios then the result is behaviourally equivalent to another partial order scenario. This lemma will be key in proving the main result for this section. It proves that irrespective of which communication semantics U we apply the result of concatenating two partial order scenarios is always another partial order scenario that is independent of U.

Lemma 4.4: Let Sc_1 and Sc_2 be two partial order scenarios. Then the sequence diagram given by $\mathsf{Pr}_U(\mathsf{Sc}_1) :: \mathsf{Pr}_U(\mathsf{Sc}_2)$ is bisimulation equivalent to a partial order scenario, which we denote as $\mathsf{Sc}_1 :: \mathsf{Sc}_2$.

Proof

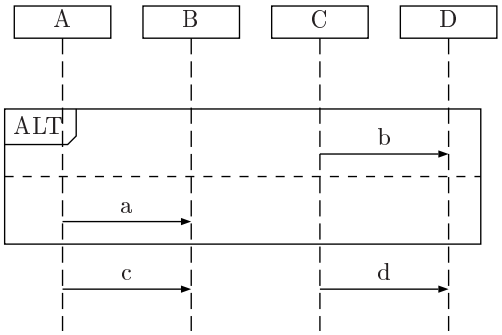

Fig. 7. Empty Alternative for Process A

Let Sd denote $\mathsf{Pr}_U(\mathsf{Sc}_1) :: \mathsf{Pr}_U(\mathsf{Sc}_2)$. Both Sc_1 and Sc_2 are defined over the same set of processes $\mathcal{P} = \{P_i \mid 1 \leq i \leq n\}$. Without loss of generality we will assume that the events for each scenario are distinct (or we can simply annotate them appropriately so we can tell which scenario they belong to). We will denote the set of events in Sc_i for process P as $E_i(P)$. The partial order over $E_i(P)$ defined by Sc_i is denoted $<^i_P$.
Define a new scenario Sc with events $E(P) = E_1(P) \cup E_2(P)$ for each $P \in \mathcal{P}$. Define partial orders $<_P$ by

- For a, $b \in E_i(P)$, $a <_P b$ if and only if $a <^i_P b$
- For $a \in E_1(P)$ and $b \in E_2(P)$ then $a <_P b$.

Clearly we have sequentially composed the process causal orders for each P. This new scenario Sc is the scenario $\mathsf{Sc}_1 :: \mathsf{Sc}_2$ referred to in the hypothesis.
Let $\phi_i = \mathsf{Pr}(In_i, S_i, <_{P_i})$, $\psi = \phi_0 \parallel_U \cdots \parallel_U \phi_n$, a string of events $\alpha = a_1 \cdots a_k$ and suppose there is a trace where

$$P_U(\mathsf{Sc}) \xrightarrow{\alpha}_\star T(B') \parallel_U \psi$$

By the construction of Sc we must have that $S_i = S^1_i \cup S^2_i$ where $S^1_i \subseteq E_1(P_i)$ and $S^2_i \subseteq E_2(P_i)$. Also we have that $In_i = In^1_i \cup In^2_i$ where $In^1_i \subseteq E_1(P_i)$ and $In^2_i \subseteq E_2(P_i)$. Let $\phi^j_i = \mathsf{Pr}(In^j_i, S^j_i, <^j_{P_i})$ for $j \in \{1, 2\}$. We also define

$$\psi^j = \phi^j_0 \parallel_U \cdots \parallel_U \phi^j_n$$

Note that the only rule that allows events to be sent from Sc_2 before any event from Sc_1 is the !Concat rule. Also if a receive event $?e$ from Sc_2 occurs before some event in

Sc_1 then $!e$ must also occur before some of the events in Sc_1. In either case $\texttt{End}(P_i, \psi^1)$ will be true for the relevant P_i.

From the definitions in Figure 5, if $\texttt{End}(P_i, \psi^1)$ holds, then InBuf will be true at that point for P_i in Sc. That implies that for all of the various communication semantics U given by Figure 5 the corresponding operational rule has a valid trigger. The converse is also true so that if we were able to execute an action in $E_2(P_i)$ for process $P_U(\mathsf{Sc})$ for any communication semantics U then $\texttt{End}(P_i, \psi^1)$ will be true. Hence we have that

$$T(\{\ \}) \parallel_U \mathsf{Sd} \xrightarrow{\alpha}_\star T(B') \parallel_U (\psi^1 :: \psi^2)$$

The converse can be shown to hold in an analogous manner. Putting all of this together gives us the bisimulation equivalence as required. □

From a sequence diagram we can define a set of partial order scenarios generated by taking a specific choice within each of the alternatives in the sequence diagram. These scenarios define a partition of the concurrent threads in the parent sequence diagram.

Definition 4.5: For a sequence diagram process term Sd*, define the set* $\mathbb{S}c(\mathsf{Sd})$ *recursively as follows:*

- $\mathbb{S}c(\mathsf{Sd}) = \{\mathsf{Sd}\}$ *when* Sd *is iteself a partial order scenario process term.*
- $\mathbb{S}c(\mathsf{Sd}_1 + \mathsf{Sd}_2) = \mathbb{S}c(\mathsf{Sd}_1) \cup \mathbb{S}c(\mathsf{Sd}_2)$
- $\mathbb{S}c(\mathsf{Sd}_1 :: \mathsf{Sd}_2) = \{\mathsf{Sc}_1 :: \mathsf{Sc}_2 \mid \mathsf{Sc}_1 \in \mathbb{S}c(\mathsf{Sd}_1) \text{ and } \mathsf{Sc}_2 \in \mathbb{S}c(\mathsf{Sd}_2)\}$
- $\mathbb{S}c(\mathsf{Sd}^\infty) = \{\mathsf{Sc}_1 :: \cdots :: \mathsf{Sc}_n \mid n \in \mathbb{N}, \mathsf{Sc}_i \in \mathbb{S}c(\mathsf{Sd}) \text{ for } 1 \leq i \leq n\}$

When $\mathsf{Sc} \in \mathbb{S}c(\mathsf{Sd})$ *we say* Sc *is included in* Sd*. From Lemma 4.4 it follows that when* Sd *is a sequence diagram then* $\mathbb{S}c(\mathsf{Sd})$ *is bisimulation equivalent to a set of partial order scenarios.*

A deadlock occurs in a sequence diagram if and only if it includes a partial order scenario that deadlocks, which we prove in Proposition 4.6. When combined with the results in Section 3, Proposition 4.6 proves that a regular sequence diagram will deadlock if and only if it contains a chase or sprint condition.

Proposition 4.6: Let X *be a sequence diagram process term and* Sd *be a sequence diagram. There is a deadlock trace*

$$T(\{\ \}) \parallel_U \mathsf{Sd} \xrightarrow{\alpha}_\star T(B) \parallel_U X$$

if and only if there exists a partial order scenario $\mathsf{Sc} \in \mathbb{S}c(\mathsf{Sd})$ *and for some* $Y \in \mathbb{S}c(X)$*,*

$$P_U(\mathsf{Sc}) \xrightarrow{\alpha}_\star T(B) \parallel_U Y$$

is also a deadlock trace.

Proof

It is straightforward to show that if there exists $\mathsf{Sc} \in \mathbb{S}c(\mathsf{Sd})$ and there is a deadlock trace $P_U(\mathsf{Sc}) \xrightarrow{\alpha}_\star T(B) \parallel_U Y$, then we also have a deadlock trace $T(\{\ \}) \parallel_U \mathsf{Sd} \xrightarrow{\alpha}_\star T(B) \parallel_U X$ for some suitable X. It therefore only remains to prove the converse.

Suppose then that we have a deadlock trace $T(\{\ \}) \parallel_U \mathsf{Sd} \xrightarrow{\alpha}_\star T(B) \parallel_U X$, where $\alpha = a_1 \cdots a_n$. To complete the proof it is enough to find $Y \in \mathbb{S}c(X)$ and some $\mathsf{Sc} \in \mathbb{S}c(\mathsf{Sd})$ where $P_U(\mathsf{Sc}) \xrightarrow{\alpha}_\star T(B) \parallel_U Y$. Without loss of generality we assume we have annotated events in Sd so that the events in each particular alternative within it can be considered distinct.

Let $E(\mathsf{Sd})$ be the set of events in a sequence diagram process term. Let $E(\alpha)$ be the events in the string α. For a sequence diagram process term Sd, define a partial order scenarios process term $(\mathsf{Sd})^\alpha$ recursively as follows:

- When Sd is a partial order scenario process term then $(\mathsf{Sd})^\alpha = \mathsf{Sd}$ if $E(\mathsf{Sd}) \cap E(\alpha) \neq \{\ \}$ otherwise $(\mathsf{Sd})^\alpha = \mathtt{End}$.
- $(\mathsf{Sd}_1 + \mathsf{Sd}_2)^\alpha = (\mathsf{Sd}_1)^\alpha$ if $E(\mathsf{Sd}_1) \cap E(\alpha) \neq \{\ \}$, $(\mathsf{Sd}_1 + \mathsf{Sd}_2)^\alpha = (\mathsf{Sd}_2)^\alpha$ if $E(\mathsf{Sd}_2) \cap E(\alpha) \neq \{\ \}$, otherwise $(\mathsf{Sd}_1 + \mathsf{Sd}_2)^\alpha = \mathtt{End}$.
- $(\mathsf{Sd}_1 :: \mathsf{Sd}_2)^\alpha = \mathsf{Sd}_1^\alpha :: \mathsf{Sd}_2^\alpha$
- $(\mathsf{Sd}^\infty)^\alpha = (\mathsf{Sd} :: \cdots :: \mathsf{Sd})^\alpha$, which consists of n copies of Sd concatenated together.

Let Sd_i be process terms so that the deadlock trace $T(\{\ \}) \parallel_U \mathsf{Sd} \xrightarrow{\alpha}_\star X$ expands as

$$T(\{\ \}) \parallel_U \mathsf{Sd} \xrightarrow{a_1} T(B_1) \parallel_U \mathsf{Sd}_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} T(B_n) \parallel_U \mathsf{Sd}_n = T(B_n) \parallel_U X$$

By Lemma 4.4 Sd^α is a partial order scenario and hence $\mathsf{Sd}^\alpha \in \mathbb{S}c(\mathsf{Sd})$. We can now

constuct a deadlock trace for Sd^{α}:

$$P_U(\mathsf{Sd}^{\alpha}) \xrightarrow{a_1} T(B_1) \parallel_U \mathsf{Sd}_1^{\alpha} \xrightarrow{a_2} \cdots \xrightarrow{a_n} T(B_n) \parallel_U \mathsf{Sd}_n^{\alpha} = T(B_n) \parallel_U Y$$

This completes the proof. □

5. CONCLUSION

Where sequence diagrams are constrained to follow MRA semantics deadlocks are not possible since coordination is always guaranteed between processes. In this paper we have considered various commonly used communication semantics, which were taken from industrial case studies at Motorola and DaimlerChrysler. For example FIFO and token ring as well as eager and lazy message consumption. We formalised these communication semantics with a process algebra that generalises the MRA semantics for regular sequence diagrams.

We refined the idea of race condition into chase and sprint conditions. For each of the semantics we considered we characterised deadlocks either in terms of sprint conditions, or in terms of chase conditions. The chase and sprint conditions together exactly determine when a deadlock can occur in a sequence diagram with one of the communication semantics that we considered.

REFERENCES

[1] R. Alur and M. Yannakakis, Model checking of message sequence charts, Proceedings of the Tenth International Conference on Concurrency Theory, LNCS 1661, Springer, pp 114–129, 1999.

[2] R. Alur, K. Etessami, M. Yannakakis, Inference of Message Sequence Charts, Proceedings 22nd International Conference on Software Engineering, pp 304-313, 2000.

[3] R. Alur, K. Etessami, M. Yannakakis, Realizability and verification of MSC graphs, Proceedings of the 28th International Colloquium on Automata, Languages, and Programming, Springer, New York, pp 797-808, 2001.

[4] D. Amyot , A. Eberlein, An evaluation of scenario notations and construction approaches for telecommunication systems development, Telecommunications Systems, v.24 n.1, pp. 61-94, 2003

[5] Paul Baker, Paul Bristow, Simon Burton, David King, Clive Jervis, Bill Mitchell, Robert Thomson, Detecting and Resolving Semantic Pathologies in UML Sequence Diagrams, Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp 50-59, 2005.

[6] P. Baker, P. Bristow, C. Jervis, D. King, B. Mitchell, Automatic Generation of Conformance Tests From Message Sequence Charts, Proceedings of 3rd SAM Workshop 2002, Telecommunications and Beyond: The Broader Applicability of MSC and SDL, pp 170-198, LNCS 2599.

[7] H. Ben-Abdhallah, S. Leue, Syntactic detection of process divergence and non-local choice in message sequence charts, Proceedings of the 3rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, pp 259-274, 1997.

[8] M. Beyer, W. Dulz, F. Zhen, Automated TTCN-3 Test Case Generation by Means of UML Sequence Diagrams and Markov Chains, Proceedings of 12th Asian Test Symposium (ATS'03), pp 102–106, IEEE 2003.

[9] Yves Bontemps, Pierre-Yves Schobbens, Synthesis of Open Reactive Systems from Scenario-Based Specifications, Third International Conference on Application of Concurrency to System Design (ACSD'03), IEEE, pp 41-50, 2003.

[10] Yves Bontemps, Patrick Heymens, Turning high-level live sequence charts into automata, Proc of Scenarios and State Machines: Models Algorithms and tools, 24th International Conf. on Software Engineering, May 2002, ACM.

[11] Sang Chung, Hyeon Soo Kim, Hyun Seop Bae, Yong Rae Kwon, Byung Sun Lee, Testing of Concurrent Programs based on Message Sequence Charts, Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems, 1999. Proceedings, pp 72 - 82, 1999.

[12] Methodological approach to the use of object-orientation in the standards making process, ETSI EG 201 872, 2001.

[13] Guidelines for the use of formal SDL as a descriptive tool, ETSI EG 202 106, 2003.

[14] T. Gehrke, M. Hilhn, H. Wehrkeim, An Algebraic Semantics for Message Sequence Chart Documents, in Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII), pp 3–18, 1998.

[15] E. Gunter, A. Muscholl, D. Peled, Compositional Message Sequence Charts, Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems table of contents, Springer LNCS 2031, pp 496 - 511, 2001.

[16] Gerard J. Holzmann, Doron A. Peled, and Margaret H. Redberg, An analyzer for message sequence charts, Software Concepts and Tools, 17(2), pp 70–77, 1996.

[17] David Harel, Werner Damm LSCs: Breathing Life into Message Sequence Charts, Formal Methods in System Design, 19, pp 45-80, 2001.

[18] David Harel, H. Kugler, Synthesizing state-based object systems from LSC specifications, International Journal of Foundations of Computer Science, 13(1), pp 5–51, 2002.

[19] D. Harel, H. Kugler, A. Pnueli, Synthesis revisited: Generating statechart models from scenario-based requirements, Formal Methods in Software and System Modeling, LNCS 3393, Springer, pp 309–324, 2005.

[20] I. Kruger, R. Grosu, P. Scholz, M. Broy, From MSCs to statecharts, Proceedings of the IFIP WG10.3/WG10.5 international workshop on Distributed and parallel embedded systems, Kluwer Academic Publishers, pp 61-71, 1998.

[21] S. Leue, L. Mehrmann, M. Rezai, M, Synthesizing software architecture descriptions from Message Sequence Chart specifications, Proceedings 13th IEEE International Conference on Automated Software Engineering, pp 192–195, 1998.

[22] M. Lohrey, Safe Realizability of High-Level Message Sequence Charts, Proceedings of the 13th International Conference on Concurrency Theory, p.177-192, 2002.

[23] S. Mauw, M. A. Reniers, An Algebraic Semanitcs of Basic Message Sequence Charts, The Computer Journal, 7(5): 473-509, 1995.

[24] S. Mauw, M. A. Reniers, Operational Semantics for MSC'96, Computer Networks, 31(17), 1785–1799, 1999.

[25] Bill Mitchell, Resolving Race Conditions in Asynchronous Partial Order Scenarios, IEEE Transactions of Software Engineering, VOL. 31, NO. 9, pp 767- 784, 2005.

[26] Bill Mitchell, Inherent Causal Orderings of Partial Order Scenarios, International Colloquium on Theoretical Aspects of Computing, Guiyang China September 2004, pp 114–129, LNCS 3407.

[27] Bill Mitchell, Robert Thomson, Clive Jervis, Phase Automaton for Requirements Scenarios, Feature Interactions in Telecommunications and Software Systems VII, 77-84, 2003, IOS Press.

[28] Object Management Group. Unified Modeling Language Specification, Version 2.0 Specification, OMG, 2004. http://cgi.omg.org/.

[29] E. Rudolph, I. Schieferdecker, J. Grabowski: Development of a MSC/UML Test Format. 153-164, Formale Beschreibungstechniken fur verteilte Systeme, pp 153-164, Verlag Shaker 2000, ISBN 3-8265-7491-5.

[30] A. Rountev, B. Connell, Object Naming Analysis for Reverse-Engineered Sequence Diagrams, Proceedings. 27th International Conference on Software Engineering, 2005, St. Louis, Missouri, USA, pp 254–263, 2005.

[31] Johann Schumann, Jon Whittle, Generating Statechart Designs From Scenarios, Proceedings of the 22nd international conference on Software engineering, pp 314-323, 2000.

[32] J. Sun, J. Song Dong, Synthesis of Distributed Processes from Scenario-Based Specifications, proceedings of FM 2005: Formal Methods, International Symposium of Formal Methods Europe, LNCS 3582, Springer, pp 415-431, 2005.

[33] H. H. Wang, S.Qin, J. Sun, J. Song Dong, Realizing Live Sequence Charts in SystemVerilog, Proceedings of First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE, IEEE, pp 379-388, 2007.

[34] J. Whittle, J. Saboo, R. Kwan, From scenarios to code: an air traffic control case study, Journal of Software and Systems Modeling, Springer, 4(1), pp 71–93, 2005.

[35] S. Uchitel, J. Kramer, J. Magee, Incremental elaboration of scenario-based specifications and behaviour models using implied scenarios, ACM Transactions on Software Engineering and Methodology (TOSEM), V13, N1, pp 37—85, 2004.

[36] S. Uchitel, G. Brunet, M. Chechik, Behaviour Model Synthesis from Properties and Scenarios, 29th International Conference on Software Engineering (ICSE'07), pp 34-43, 2007.

[37] Z.100 (11/99) ITU-T Recommendation - Languages for telecommunications applications - Specification and description language

[38] MOST Dynamic Specification, ©Most Cooperation 2005, http://www.mostcooperation.com/downloads/Specifications/MOST Specifications/

[39] Z.120 (11/99) ITU-T Recommendation - Message Sequence Chart (MSC)

[40] ITU-T Recommendation Z.120 (04/1998) Annex B - Formal semantics of Message Sequence Charts.