Inherent Causal Orderings of Partial Order Scenarios

Bill Mitchell

w.mitchell@surrey.ac.uk,
Department of Computing, University of Surrey
Guildford, Surrey GU2 7XH, UK

Abstract. Scenario based requirements specifications are the industry norm for defining communication protocols. Basic scenarios captured as UML sequence diagrams, Message Sequence Charts (MSCs) or Live Sequence Charts (LSC) have partial order semantics that characterize system traces by restricting the possible order of events within those traces. The semantic partial order of the scenario specification is called the causal ordering.
Semantic inconsistencies often occur in partial order scenarios between the specified causal ordering and the order that events can occur in practice. Such inconsistencies are known as race conditions. The paper proves that there is a unique race free partial order that is a minimal weakening of the causal ordering. In other words, there is a canonical generalization of the requirements that corrects all race conditions. Hence any race free generalization of the original scenario is in fact a generalization of the canonical scenario. The paper also proves the dual result, there is a unique race free partial order that is a minimal strengthening of the causal order. I.e. there is a canonical refinement of the requirements that corrects all race conditions.

1 Introduction

UML sequence diagrams [19], Message Sequence Charts (MSCs) [18], and Live Sequence Charts (LSCs) [7] are popular for defining wireless and mobile communication protocols. The semantics of a basic scenario diagram defined with any of these languages can be given in terms of a partial order on the events in the scenario. The partial order restricts the order in which events can occur in any system trace. This partial order is called the causal ordering. We refer to any basic scenario diagram with such a semantics as a partial order scenario.

Although scenario specification languages have become quite sophisticated and have expressive powers beyond partial order scenarios, such scenarios are still the mainstay of industrial specifications. Consequently the study of partial order scenarios is still an active topic of research [5, 13, 12, 16]. Synthesizing various types of system models directly from these partial order scenarios is also an active area [1, 3, 4, 11, 15, 17]. Research into automatic test generation from partial order scenarios is another active research area [2, 6, 14].

Industrial requirements specifications often contain inconsistencies between the specified causal ordering and the order that events can occur in practice. Race conditions are amongst the most common of these inconsistencies. Essentially a race condition asserts a particular order of events will occur as a consequence of the causal ordering, when in practise this order can not be guaranteed to occur. See [9, 10] for the original formal description of the problem within the MSC context.

It is possible to directly analyze the causal ordering to automatically detect race conditions [10]. This still leaves the onerous task of actually correcting the race conditions. Case studies such as [20] have shown that around a third of significant defects in SDL specifications are caused by poor requirements specifications. Since many SDL specifications are refined from MSCs and UML sequence diagrams this suggests a significant number of errors arise because of poor quality in partial order scenarios. Hence the ability to automatically correct race conditions would be of practical value.

In the paper we prove that given a causal ordering there exists a unique minimal weakening of that order which does not contain any race conditions, and which is itself the causal ordering of some scenario (Theorem 10). We call this weakening the inherent causal ordering, and the scenario to which it corresponds the inherent causal scenario. We prove the inherent causal scenario is canonical up to simulation equivalence of system behaviour. Therefore any race free generalization of the original scenario must be a generalization of the inherent causal scenario. Hence there is an optimal generalization of a partial order scenario that corrects all race conditions. In section 7 we describe an example MSC scenario from an industrial case study that illustrates how the inherent causal order can be of value in practise.

The paper also proves that there is a unique minimal strengthening of a causal order that corrects all race conditions, and which is equivalent to the causal ordering of some scenario (Theorem 18). We call this the inherent refinement ordering. As might be expected we prove the inherent refinement scenario is canonical up to simulation equivalence. Hence there is an optimal refinement of a partial order scenario that corrects all race conditions. The results can be generalized to scenarios that extend the basic partial order semantics with iteration and branching, as is the case with HMSCs. However, we do not prove that here due to lack of space.

Although our results are perfectly general and apply to any basic scenario diagram language such as basic UML sequence diagrams, MSCs or LSCs, we will use MSC as the central language for the paper. The MSC standard [18] is stable and MSCs are common in industry. Also MSC 2000 is being adopted within UML 2.0 [19]. In addition MSCs allow the most general form of causal ordering since it is possible for an MSC causal order to be almost any irreflexive transitive partial order.

2 Basic Partial Order Specifications

In this section we define the causal ordering semantics for partial order scenarios (e.g. basic MSCs). We use the same message semantics as the MSC 2000 standard [18]. Hence, a partial order scenario defines a set of message exchanges between processes with asynchronous communication channels. Also we do not assume any type of buffering with the channels. However, the results in the paper do hold for both synchronous and FIFO channels.

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. We regard $!m$ as belonging to the sending process, and $?m$ as belonging to the receiving process. Let E be the set of all send and receive events between all processes. Each event has a label, let $l : E \longrightarrow L$ be the labelling function. For a message m, $l(!m) = l(?m)$. Within the MSC standard there are many other kinds of events such as action boxes and condition symbols, but here we only consider message events to simplify proofs as much as possible. It is straightforward to generalize the results to include these other events.

Definition 1. *A partial order scenario on processes $\mathcal{P}$ is*

- *a collection of disjoint sets $E(P) \subseteq E$, for each $P \in \mathcal{P}$ that defines the message events belonging to P,*
- *and a set of irreflexive partial orders $<_P$, where $<_P$ is a partial order on $E(P)$ that defines the local ordering of events for process P.*

These local partial orders must be 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 messages are allowed to be sent from a process to itself, so we allow $P = Q$. 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 that represent the relevant order relations. It is important to note the local orders are not necessarily total, but can be any irreflexive partial order. In the literature it is sometimes assumed basic scenario diagrams have total local orderings, so it is worth emphasizing this does not have to be the case.

Let Msg be the set of messages defined as the set of send and receive event pairs:

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

Definition 2. *The causal ordering $<_{\mathcal{C}}$ on a partial order scenario is the transitive closure of the relation given by*

$$\left(\bigcup_{P \in \mathcal{P}} (<_P) \right) \cup \mathsf{Msg}$$

From now on we will assume that all partial orders are transitive and irreflexive without loss of generality. We will also assume that all causal orderings are

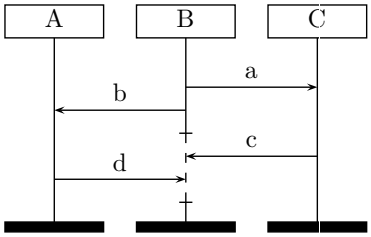

Fig. 1. Race hidden by coregion

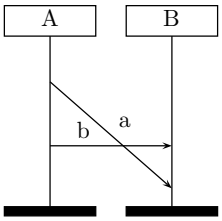

Fig. 2. Message Overtaking, which is prohibited

irreflexive, so that messages must be received after they are sent. We also, as is the norm, rule out message overtaking as shown in figure 2. The MSC standard includes a general ordering construct, which is a simple graphical notation that explicitly forces one event to occur before another event in the causal order. A general order construct is depicted as a dashed arrow between the events to be ordered, with arrow head placed in the middle of the arrow. In combination with the coregion construct that means a process order $<_P$ defined by an MSC can be any arbitrary irreflexive transitive partial order on the events $E(P)$.

The causal ordering defines the set of all possible system traces that are given by the partial order scenario. A system trace is any total order extension of $<_{\mathcal{C}}$. Recall a total order on a set S is a partial order $<$ on S where for any distinct elements x, $y \in S$, either $x < y$ or $y < x$.

Definition 3. *The set of system traces defined by a causal ordering $<_{\mathcal{C}}$ is the set of total order extensions of $<_{\mathcal{C}}$.*

Consider the MSC depicted graphically in figure 1. Each vertical line describes the time-line for a process, where time increases down the page. 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. Within a coregion events are not locally ordered unless that is directly imposed by a general order construct. Coregions are depicted with a dashed line. For process B events $?c$ and $?d$ are unordered as they occur within a coregion. The local partial orders defined by this MSC are given in figure 3 where we draw

the ordering downwards, so that $!a <_B !b$ for example. In this case the causal ordering $<_\mathcal{C}$ is given in figure 4.

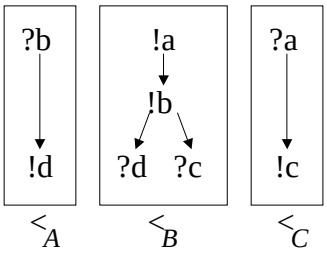

Fig. 3. Process Partial Orders

Figure 1 illustrates a race condition. The causal ordering asserts that $!b <_\mathcal{C} ?c$. If this MSC is taken as a specification it asserts that after C receives a it must send c so that it arrives after b is sent. It is not possible for C to know for sure when $!b$ occurs without querying B. Hence it is quite possible if this scenario is implemented naively that c will arrive before b is sent, contradicting the specification. This error can occur even though each of the processes A and C locally implements the specification correctly.

Definition 4. *Define a partial order $<$ on E to be race free when for every event x and message e:*

$$x <?e \Rightarrow (x <!e \text{ or } x =!e)$$

We define an MSC to be race free when its causal ordering is race free.

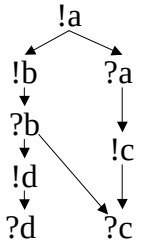

Fig. 4. Causal Ordering

That is $<$ is race free if the following holds. When $<$ orders an event x before the receive event of some message e, then it also orders x to be before the send event of e.

Note that figure 1 is not race free since $!b <_\mathcal{C} ?c$, but $!b \not<_\mathcal{C} !c$. The three basic types of race are illustrated in figure 5. In the first example we have $?a <_\mathcal{C} ?b$,

but $\neg(?a <_{\mathcal{C}} !b)$. In the second example we have $!a <_{\mathcal{C}} ?b$, but $\neg(!a <_{\mathcal{C}} !b)$. In the last example we have $?a <_{\mathcal{C}} ?c$, but $\neg(?a <_{\mathcal{C}} !c)$. The third race example is the only one of the three that can be avoided by forcing messages to be synchronous. Hence the first two examples will cause semantic inconsistencies in synchronous and asynchronous scenarios.

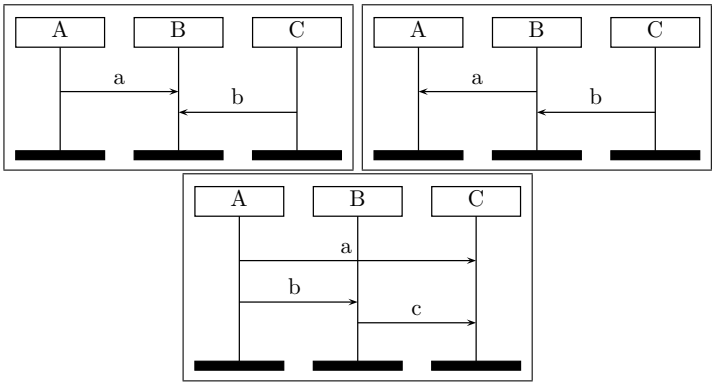

Fig. 5. Three basic types of race condition

3 Partial Order Processes

In this section we define a process algebra term that characterizes the traces that are defined by a causal ordering. This is a standard result for partial orders, but we present it in a slightly non-standard format for ease of use later in the paper. The process algebra term also characterizes the system behaviour up to strong bisimulation equivalence.

First we set up some notation for defining sets of events that are important in generating system and process traces. Let $<$ be a partial order on a set of events E. For a set $S \subseteq E$ define

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

The set $\mathsf{n}(S,<)$ are those events that are a least upper bound for some element in S. The set $\mathsf{m}(S,<)$ is just the set of minimal elements of S.

The set $\mathsf{af}(a,S,<)$ characterizes how events may follow a in an execution trace, where S describes the set of all events that are eligible to occur concurrently with a. Suppose we have an execution 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 in t_1. Then t_1 must be of the form $b \cdot t_2$ where $b \in \mathsf{af}(a, S, <)$. The first element that can occur in a trace that is a total extension of $<$ has to come from $\mathsf{m}(E, <)$. Hence we can define the system behaviour for a causal ordering as follows.

Definition 5. *For a set $S \subseteq E$ define a recursive process algebra term by*

$$P(S, <) = \sum_{\{a \in S\}} a \cdot P(\mathsf{af}(a, S, <), <)$$

and $P(\emptyset, <) = 0$.

Where $a \cdot P$ denotes the usual sequential composition of action and process, and the summation is nondeterministic choice (as standard in both CCS and CSP).

Definition 6. *Define the observable behaviour for causal ordering $<_{\mathcal{C}}$ to be the process:*

$$P(M) = P(\mathsf{m}(E, <_{\mathcal{C}}), <_{\mathcal{C}})$$

In [8] they define a process algebra term for M that defines the system traces for $<_{\mathcal{C}}$. Let $P^*(M)$ denote this process. Process $P(M)$ is strong bisimulation equivalent to $P^*(M)$. Hence $P(M)$ defines the system traces of the global behaviour for the processes defined by M.

Suppose that MSC M contains processes P_i for $1 \leq i \leq n$. Then a parallel composition of the $P(\mathsf{m}(E(P_i), <_{P_i}), <_{P_i})$ for $1 \leq i \leq n$ is strong bisimulation equivalent to $P(M)$ (which follows from an analogous result in [8]). However, we will not need to use that result here.

4 Inherent Causal Behaviour

A partial order $<$ on events preserves the message ordering when $!e <?e$ for every message e. Let $<_{\mathcal{C}}$ be the causal ordering for a partial order scenario.

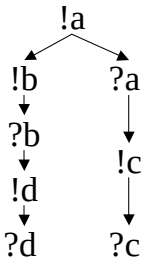

Fig. 6. Inherent Ordering

Definition 7. *Define the inherent causal ordering* $<_I$ *of* $<_\mathcal{C}$ *to be the transitive closure of the following partial order relation* $<$*. For every event* x *and message* e *define:*

1. $x < !e \iff x <_\mathcal{C} !e$
2. $!e < ?e$

Note that when regarding a partial order as a set of pairs, we have

$$(<_I) \subseteq (<_\mathcal{C})$$

The inherent ordering is the causal order of some partial order scenario. This follows from the next lemma.

Lemma 8. *The inherent causal ordering* $<_I$ *of a partial order scenario with processes* $\mathcal{P}$ *is the transitive closure of the following partial order relation.*

1. $x < !e \iff \exists P \in \mathcal{P}$ *such that* $x <_P !e$
2. $!e < ?e$

Figure 6 gives a graphical depiction of the inherent ordering for figure 1.

Since we are able to impose general orderings on events within MSC diagrams we can represent this inherent ordering as an MSC. That is we can define a second MSC who's causal ordering is in fact the inherent ordering of figure 1. This is the leftmost MSC in figure 7. Notice that the coregion for process B now covers all the events in $E(B)$. In order to assert that $!a$ must occur before $!b$ we have added a general ordering construct between these events. This is the dashed arrow, with arrow head placed at the mid point of the arrow. Wherever such a general ordering arrow occurs in an MSC from events x to y this explicitly defines $x <_\mathcal{C} y$. Thus definition 2 of $<_\mathcal{C}$ has to be extended so that it includes the set of pairs given by the general ordering construct.

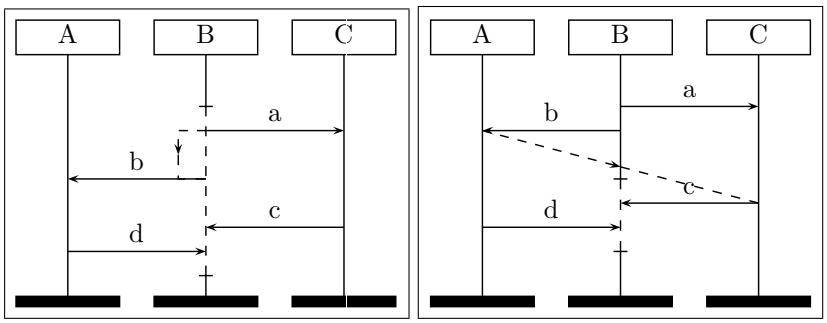

Fig. 7. Inherent Causal Ordering, and Inherent Refinement Ordering as MSCs

5 Canonical Inherent Processes

Recall in section 3 we defined the observable process behaviour $P(M)$ of a partial order scenario M.

Definition 9. *Define the inherent process behaviour of a partial order scenario M to be $P_I(M) = P(\mathsf{m}(E, <_I), <_I)$*

Let $\sqsupset$ denote the standard simulation relation for process algebras. That is $P \sqsupset Q$ iff

for every transition $Q \xrightarrow{a} Q'$,
there exists a transition $P \xrightarrow{a} P'$ where $P' \sqsupset Q'$

Theorem 10.

- $(<_I) \subseteq (<_{\mathcal{C}})$, *and* $P_I(M) \sqsupset P(M)$
- *For any race free partial order $<$ that preserves message ordering, let* $P_< = P(\mathsf{m}(E, <), <)$.
  Then $P_< \sqsupset P(M)$ *iff* $(<) \subseteq (<_I) \subseteq (<_{\mathcal{C}})$ *and* $P_< \sqsupset P_I(M)$

That is $P_I(M)$ is the canonical process that simulates $P(M)$ and is race free. To say that $(<_1) \subseteq (<_2)$ means that for every x and y in E, when $x <_1 y$ then $x <_2 y$.

This theorem proves that the order $<_I$ describes the maximal ordering with respect to simulation equivalence that is a race free weakening of $<_M$. Hence constructing an MSC that has partial order semantics given by $<_I$ defines a new MSC that corrects any race conditions in M, and weakens the causal ordering of M as little as possible. It is straightforward to construct such an MSC.

The theorem is a consequence of the following lemmas. For any partial order $<$ (which is not necessarily race free) let $T(<)$ be the set of total extensions of $<$. Lemma 11 follows immediately from our initial observations concerning the definition of $\mathsf{af}(a, S, <)$.

Lemma 11. *The set of traces of $P_<$ is exactly $T(<)$, the set of total order extensions of $<$.*

Lemma 12. *For partial orders $<_1$ and $<_2$ where*

$$P_{<_1} \sqsupset P_{<_2}$$

then $(<_1) \subseteq (<_2)$

Proof

Note that $x < y$ iff for every trace in $T(<)$, x occurs before y in the trace. When $P_{<_1} \sqsupset P_{<_2}$ then the set of traces for $P_{<_2}$ is contained in the set of traces for $P_{<_1}$, that is $T(<_2) \subseteq T(<_1)$.

$x <_1 y \Rightarrow x$ occurs before y in every trace of $T(<_1)$
$\Rightarrow x$ occurs before y in every trace of $T(<_2)$
$\Rightarrow x <_2 y$

Hence $(<_1) \subseteq (<_2)$, which concludes the proof.

Lemma 13. *Given two partial orders $<_1$ and $<_2$,*
$T(<_1) \subseteq T(<_2)$ iff $P_{<_1} \sqsubset P_{<_2}$

Proof

Note that $T(<_1) \subseteq T(<_2)$ iff $(<_2) \subseteq (<_1)$. Given $(<_2) \subseteq (<_1)$, to prove $P_{<_1} \sqsubset P_{<_2}$, it is enough to prove that for any $S \subseteq E$ and $a \in E$,

$$\mathsf{af}(a, S, <_1) \subseteq \mathsf{af}(a, S, <_2) \tag{1}$$

Let

$$m_1 = \mathsf{m}((S - \{a\}) \cup \mathsf{n}(\{a\}, <_1), <_1)$$
$$m_2 = \mathsf{m}((S - \{a\}) \cup \mathsf{n}(\{a\}, <_2), <_2)$$

We write $U \leq V$ for sets $U, V \subseteq E$, when for each $u \in U$, there is some $v \in V$ such that $u \leq v$. Note that since $(<_2) \subseteq (<_1)$ then $\mathsf{n}(\{a\}, <_2) \leq \mathsf{n}(\{a\}, <_1)$.

For a contradiction suppose that $x \in m_1$ and $x \notin m_2$. This implies there is some $y \in m_2$ such that $x <_2 y$. First consider if $y \in S - \{a\}$. Then $x <_1 y \in S - \{a\}$, hence $x \notin m_1$. This is a contradiction, hence we must have $y \in \mathsf{n}(\{a\}, <_2)$. Since $\mathsf{n}(\{a\}, <_2) \leq \mathsf{n}(\{a\}, <_1)$, there is some $y' \in \mathsf{n}(\{a\}, <_1)$ such that $x <_2 y <_1 y'$. Therefore $x <_1 y' \in \mathsf{n}(\{a\}, <_1)$, and so $x \notin m_1$. Again a contradiction as required to complete the proof of equation 1. The proof that $T(<_1) \subseteq T(<_2)$ implies $P_{<_1} \sqsubset P_{<_2}$, is completed once we note that $\mathsf{m}(E, <_1) \subseteq \mathsf{m}(E, <_2)$.

The converse implication is straightforward. It is true for any processes P and Q that if $P \sqsupset Q$ then the set of traces of Q is contained in the set of traces for P. Since the traces of $P_{<_i}$ are exactly $T(<_i)$, the result is then immediate. That completes the proof of the lemma. □

Lemma 14. *For a partial order $<$ that preserves message ordering and is race free,*

$$\Big((<) \subseteq (<_C)\Big) \Rightarrow \Big((<) \subseteq (<_I) \subseteq (<_C)\Big)$$

Proof

For this it is enough to prove that whenever $x < y$ then $x <_I y$. The proof splits into cases depending on whether y is a receive or send event. First suppose that $y = !e$ for some message e. Then $x <!e$ implies $x <_C !e$ since $(<) \subseteq (<_C)$. By definition of $<_I$, $x <_C !e$ implies $x <_I !e$.

The other case is where $y = ?e$ for some message e. Since $<$ is race free, $x <?e$ implies that $x <!e$. As above this implies $x <_I !e$. The ordering $<_I$ preserves message ordering, and hence $x <_I ?e$. This completes the proof of the lemma. □

6 Inherent Refinement Behaviour

In this section we prove the dual result of theorem 10, where instead of generalizing the causal ordering we refine it.

Definition 15. *Define the inherent refinement ordering $<_R$ of a causal ordering $<_\mathcal{C}$ to be the transitive closure of the following partial order $<$. For every event x and message e define:*

- $x <!e \iff x <_\mathcal{C} ?e$
- $!e <?e$

First note that $<_R$ is race free. Since it is clear from the definition that $x <_R ?e$ implies that $x <_R !e$ or $x = !e$. Also notice that the refinement order only extends $<_\mathcal{C}$ by forcing particular send events to be delayed so that other events may occur first, and hence is implementable.

If the partial order scenario is an MSC then the inherent refinement ordering can be constructed by adding suitable general orderings to the MSC, which cause appropriate send events to wait until the relevant receive events have occurred. For example the rightmost MSC of figure 7 gives the inherent refinement order for the partial order scenario in figure 1.

The use of general orderings is acceptable so long as they can be implemented in asynchronous distributed systems. We only use them to delay send events, and this effect can always be achieved by adding further messages to the partial order scenario. To force a general ordering $x <_\mathcal{C} !e$, where $x \in E(P)$ and $!e \in E(Q)$, we can add a new coordination message c with $!c \in E(P)$ and $?c \in E(Q)$. We then alter the local orderings by adding the pairs $x <_P !c$ and $?c <_Q !e$. Processes P and Q can enforce these orderings locally since they only have to delay send events to do so.

However, the choice of implementation is for the system designers who may use other mechanisms which are more appropriate for their particular circumstances. The goal of the solution presented here is to correct the semantics for the scenario in an optimal manner without altering the given message content of the scenario and without imposing any assumptions about communication channel semantics.

Lemma 16.

$$(<_\mathcal{C}) \subseteq (<_R)$$

Proof

To prove this suppose $x <_\mathcal{C} y$. If $y = !e$ for some e, then $y <_\mathcal{C} ?e$. Hence from the definition $x <_\mathcal{C} ?e$ and hence $x <_R !e$. That is $x <_R y$.

When $y = ?e$, then $x <_R !e$. Also $!e <_R ?e$, hence by transitive closure, $x <_R ?e = y$. This completes the proof of the lemma. □

Lemma 17. *For any race free transitive partial order $<$ that preserves messages and where $(<_\mathcal{C}) \subseteq (<)$, then*

$$(<_\mathcal{C}) \subseteq (<_R) \subseteq (<)$$

Proof

To prove this first consider an event x and message e where $x \neq ?e$ and $x <_{\mathcal{C}} ?e$. That is $x <_R !e$. Since $(<_{\mathcal{C}}) \subseteq (<)$, we have $x <?e$. Since $<$ is race free we have $x <!e$. Hence, if $x <_R !e$ then $x <!e$. Since $<$ preserves messages it trivially follows that $!e <_R ?e$ implies $!e <?e$. Hence as $<$ is transitive we have proved that $(<_R) \subseteq (<)$. □

Given the lemmas already proved in section 5 we have thus proved the following theorem, which is the dual to theorem 10.

Theorem 18. *Let* $P_R(M) = P(\mathsf{m}(E, <_R), <_R)$*, then*

- $(<_{\mathcal{C}}) \subseteq (<_R)$*, and* $P(M) \sqsupset P_R(M)$
- *For any race free partial order* $<$ *that preserves message ordering, let* $P_< = P(\mathsf{m}(E, <), <)$.
  Then $P(M) \sqsupset P_<$ *iff* $(<_{\mathcal{C}}) \subseteq (<_R) \subseteq (<)$ *and* $P_R(M) \sqsupset P_<$

Hence $<_R$ is the canonical refinement of the causal order that corrects all race conditions in the specification.

7 Industrial Case Study Example

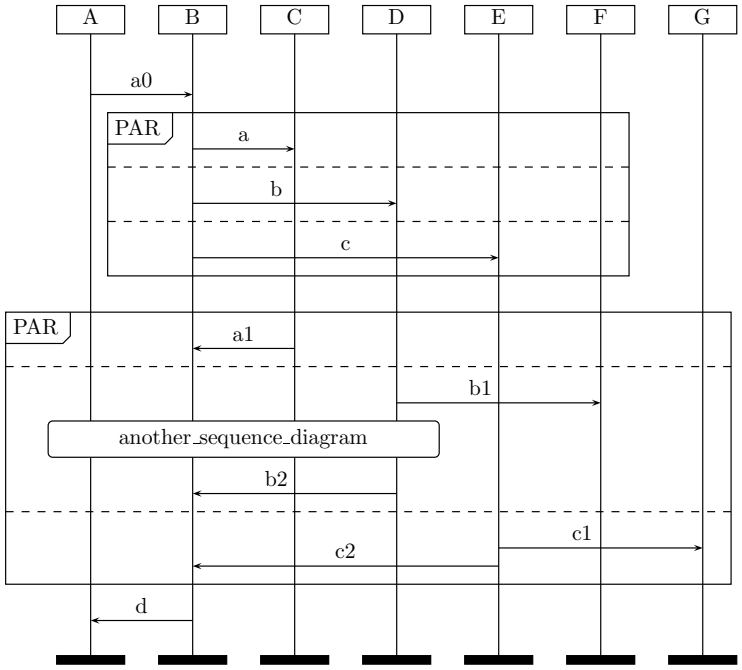

Fig. 8. UML 2.0 case study example with multiple race conditions

In collaboration with Motorola Research Labs, we have been conducting a number of case studies [6, 11] into automating pathology detection in MSC telecommunication specifications. Figure 8 is an anonymized example from a Motorola case study, which contains multiple race conditions. The original diagram is a UML 2.0 sequence diagram that describes traffic channel allocation and activation between various processes for a telecommunication protocol. Process A has delegated the task of determining what resource to allocate to process B.

A parallel construct in a MSC/UML sequence diagram, denoted by `PAR`, describes a set of concurrent threads that occur in the diagram. Dotted lines delineate the different threads. Hence, events from one thread are not causally ordered with respect to events from any other thread. Figure 8 contains two parallel constructs. The first parallel construct contains messages a, b and c in separate threads, which can therefore occur in any order. The bounding box of a parallel construct has no effect on the ordering of events, it solely delineates the scope of the concurrent threads. Note an MSC/UML sequence diagram containing solely messages, coregions and parallel constructs still defines a partial order scenario in the sense of definition 1.

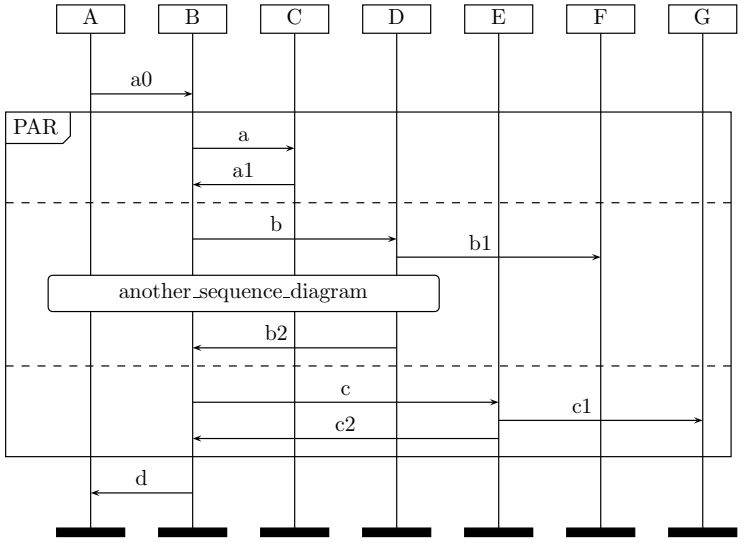

Fig. 9. Inherent Causal Scenario for Figure 8 as MSC

An inline reference, denoted by `REF`, is a place holder for another sequence diagram. The reference can be replaced by the contents of the other sequence diagram if desired. The reference is weakly composed with the referring diagram when inlined. Figure 8 contains an inline reference spanning processes A through

D. We will assume for this example that the inline reference is some linear ordering of events in order to simplify our calculations.

In total we have the following six race conditions in figure 8. Event $?a1$ is in a race with $!b$ and also with $!c$. Event $?c2$ is in a race with $!a$ and also with $!b$. Also event $?b2$ is in a race with $!a$ and also with $!c$. It may be that the authors implicitly assumed the downlink latency from B is much shorter than the uplink latency for the other processes. If this were true it may be possible in practise for the specification to be realizable. However it is far safer to rewrite the specification without these race conditions.

One way to remove these races would be to regroup the messages within a single parallel construct. Messages a and $a1$ could be grouped within the same thread of a parallel construct. Similarly b, $b1$, $b2$ and the inline reference could be grouped in a second thread. Finally c, $c1$ and $c2$ could be grouped in the third thread. Figure 9 depicts this solution. It seems reasonable to suppose this will not contradict what the authors originally intended.

Figure 9 is exactly the inherent causal scenario of figure 8. In this case the inherent causal order for figure 8 would seem to represent the specification intended by the authors, rather than the causal order of figure 8 itself.

The UML 2.0 case study also contained cases of sequence diagrams where a more intuitively 'correct' specification was given by the inherent refinement ordering, rather than the inherent causal ordering. Hence, we are not proposing either inherent ordering as some kind of panacea. However, providing practitioners with both inherent orderings in the form of MSCs (or UML sequence diagrams) will give them a better understanding of what specifications are possible.

8 Conclusion

The paper has proved that there is a canonical solution for correcting all race conditions within a partial order scenario by weakening the causal relationship. Moreover the solution can be easily automated via lemma 8. The inherent causal ordering that defines the solution can also be presented in MSC format by use of the MSC coregion, parallel and general ordering constructs. Section 7 gave an example from an industrial case study where the inherent causal ordering captured the intended behaviour of a specification containing multiple race conditions.

The paper has also proved the dual result, that there is a canonical refinement of the specifications that corrects all race conditions. This is the inherent refinement ordering. This also can be presented in MSC format, which can be constructed automatically. Together these inherent orderings provide a useful insight into the semantically consistent specifications that are possible for a distributed system.

References

1. R. Alur, K. Etessami, M. Yannakakis, Inference of Message Sequence Charts, Proceedings 22nd International Conference on Software Engineering, pp 304-313, 2000.
2. 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), IEEE 2003.
3. Yves Bontemps, Pierre-Yves Schobbens, Synthesis of Open Reactive Systems from Scenario-Based Specifications, (ACSD'03)
4. 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.
5. E. Gunter, A. Muscholl, D. Peled, Compositional Message Sequence Charts, TACAS 2001.
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, The Broader Applicability of MSC and SDL, pp 170-198, LNCS 2599.
7. David Harel, Werner Damm LSCs: Breathing Life into Message Sequence Charts, Formal Methods in System Design, 19, 45-80, 2001
8. T. Gehrke, M. Hilhn, H. Wehrkeim, An algebraic semantics for message sequence chart documents. In FORTE/PSTV'98, pages 3-18. Kluwer Academic Publishers, 1998.
9. Gerard J. Holzmann, Doron A. Peled, Message Sequence Chart Analyzer, United States Patent, 5,812,145.
10. Gerard J. Holzmann, Doron A. Peled, and Margaret H. Redberg, An analyzer for message sequence charts, Software Concepts and Tools, 17(2), 1996.
11. B. Mitchell, R. Thomson, C. Jervis, Phase Automaton for Requirements Scenarios, Feature Interactions in Telecommunications and Software Systems VII, 77-84, 2003, IOS Press.
12. A. Muscholl, D. Peled: From Finite State Communication Protocols to High-Level Message Sequence Charts. ICALP 2001, 720-731.
13. D. Peled: Specification and Verification using Message Sequence Charts. Electronic Notes in Theoretical Computer Science 65, No 7, 2002.
14. E. Rudolph, I. Schieferdecker, J. Grabowski: Development of a MSC/UML Test Format. 153-164, Formale Beschreibungstechniken fur verteilte Systeme, 2000. Verlag Shaker 2000, ISBN 3-8265-7491-5.
15. J. Schumann, J. Whittle, Generating Statechart Designs From Scenarios, Proceedings 22nd international conference on on Software engineering, 2000.
16. A. Tsiolakis, Integrating Model Information in UML Sequence Diagrams, Electronic Notes in Theoretical Computer Science, June 2001.
17. S. Uchitel, J. Kramer, J. Magee, Synthesis of Behavioral Models from Scenarios, IEEE Transactions on Software Engineering, vol. 29, no. 2, February 2003
18. Z.120 (11/99)ITU-T Recommendation - Message Sequence Chart (MSC)
19. Object Management Group (OMG), *Unified Modeling Language (UML): Superstructure, Version 2.0*, 2003. Available from http://www.omg.org.
20. E. Wong, J. R. Horgan, W. Zage, D. Zage and M. Syring, Applying Design Metrics to a Large-Scale Software System, (Motorola), Proceedings of the 9th International Symposium on Software Engineering Reliability (ISSRE '98), Paderborn, Germany, November 4-7, 1998.