Race Conditions in Message Sequence Charts

Chien-An Chen, Sara Kalvala, and Jane Sinclair

Department of Computer Science,
University of Warwick,
Coventry CV4 7AL, UK
{cssdc, sk, jane}@dcs.warwick.ac.uk

Abstract. Message Sequence Charts (MSCs) are a graphical language for the description of scenarios in terms of message exchanges between communicating components in a distributed environment. The language has been standardised by the ITU and given a formal semantics by means of a process algebra. In this paper, we review a design anomaly, called race condition, in an MSC specification and argue that the current solution correcting race conditions is too weak when implementation is considered. In this paper, we provide an algorithm on partial orders as our solution. The result is a strengthened partial order, which is race-free and remains race-free in the implementation.

1 Introduction

Message Sequence Charts (MSCs) [12] are a trace language, describing scenarios where messages are exchanged between communicating entities in a distributed environment. The language was designed to supplement SDL [18] by providing a graphical representation of behavioural aspects in an SDL specification. The formalism has been recommended as a standard by the ITU (International Telecommunication Union). During the last decade, it has evolved incrementally from a plain message exchange diagram to a multi-layered documentation methodology with rich constructs. Due to their readability and tool support, MSCs have become a specification language in their own right and have been enjoying a widespread use in specifying telecommunication protocols and reactive systems, particularly system requirements during early stages of development.

In addition to their industrial popularity, MSCs have been drawing much attention from researchers. Attempts at a formal semantics of MSCs have emerged since MSC'92 [6]. Various approaches have been adopted for this task, such as automata theory [13,14], Petri Nets [10], streams [5] and process algebra [7,9,15]. Synthesising system models or behavioural models from MSC scenarios is also an active topic [1,19,20]. MSC specifications can be syntactically analysed for a variety of design anomalies, such as deadlocks, race conditions [2], process divergence and non-local branching choices [3].

In this paper, we are concerned only with race conditions and their solutions. The semantics of an MSC used here, called causal ordering, is actually a partial order characterising execution traces on communication events. A race condition

K. Yi (Ed.): APLAS 2005, LNCS 3780, pp. 195–211, 2005.

refers to an inconsistency between the causal ordering specified in an MSC and the ordering that may occur in practice. The current solution [16] to correcting race conditions asserts that given a causal ordering there exists a unique race-free partial order that is a minimal weakening of the causal ordering. Dually, there also exists a unique race-free partial order that is a minimal strengthening of the causal ordering. Both partial orders have syntactically legal MSCs to match.

Here we focus on the strengthening counterpart of a causal ordering. We have observed that if enforcing the strengthened ordering properties by adding extra messages as acknowledgements, new race conditions may occur. In this situation, more acknowledgement messages are required, so that a simple MSC diagram can be flooded with unnecessary messages and becomes hard to read and analyse. Such an observation motivates our work. In this paper, we propose an algorithm on partial orders as the solution. In addition, justification on the algorithm is also provided before we implement the approach for tool-support. The result of our work shows that for a causal ordering, there exists a minimally strengthened partial order, that is race-free and remains race-free when acknowledgement messages are added to enforce the strengthened ordering properties. Our approach enables specifiers to illustrate explicitly more design details in an MSC scenario without the risk of race conditions. Although we use MSCs as the central language in this paper, our approach can be applied to other languages with partial-order semantics, e.g. UML Sequence Diagrams [4] and LSCs [8].

This paper is organised as follows. Section 2 gives an overview of the MSC language we use in this paper and its semantics. The concept of race conditions and the current solutions are also introduced. In section 3, we present the remaining problems which motivate our work; in section 4, we propose an approach to correcting race conditions without introducing new ones when acknowledgement messages are added. Conclusions are presented in section 5. Familiarity with the theories of relations and partial orders is presumed.

2 Preliminary

2.1 Message Sequence Charts

A *basic* MSC (bMSC) is a building block for an MSC document. As can be seen in Fig.1(a), $MSC1$ contains three *instances* (or *processes*), namely $P1$, $P2$ and $P3$, denoted by vertical axes. Three *messages*, a, b and c, denoted by arrows, are exchanged between those instances. The frame around the diagram represents the *environment*. $MSC1$ intuitively describes a scenario where $P1$ sends message a to $P2$; after receiving, $P2$ sends messages b and c to $P3$ and $P1$ respectively.

A temporal ordering is defined along each vertical axis and horizontal arrow in the sense that (1) the events along an instance axis proceed from top to bottom, and (2) a message must be sent before it is received. In the bMSC convention, it is assumed that the communication medium between distributed processes is reliable, i.e. no message gets lost. Furthermore, the setting of a bMSC is assumed to be of *asynchronous* communication. Hence in $MSC1$ of Fig.1(a), the order of receiving b and c is undefined. The ordering properties can be strengthened

by a *general ordering* construct, denoted by a dashed line with an arrowhead in the middle. A general ordering symbol is attached to the events that need to be ordered. As illustrated in $MSC2$ of Fig.1(b), the general ordering adds a constraint that receiving c has to occur after receiving b.

The core subset of bMSCs we use is described in the ITU-standard [12]. Our definition is in a similar style to those that can be found in the early MSC-related work, e.g. [1,2]. The bMSC notation used in this paper is defined as follows.

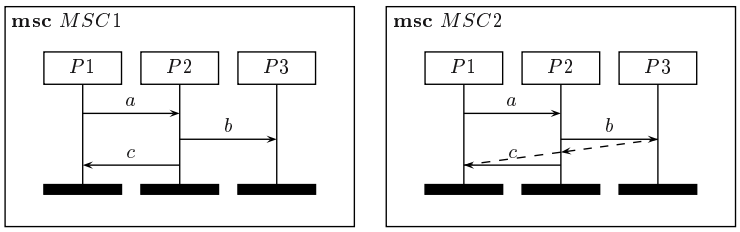

Fig. 1. (a)A bMSC diagram (b)A bMSC with a general ordering

Definition 1. *A bMSC is defined as a tuple* $\langle \mathcal{P}, \mathcal{M}, msg, \mathcal{O}, <_C \rangle$ *where:*

- $\mathcal{P}$ *is a finite set of instances, i.e.* $\mathcal{P} = \{Pi \mid i \in 1..n\}$.
- $\mathcal{M}$ *is a finite set of message names. A set* Σ *for all the events in a bMSC can be derived accordingly as* $\Sigma = \Sigma_{out} \cup \Sigma_{in}$ *where* $\Sigma_{out} = \{!a_{ij} \mid i, j \in 1..n \wedge a \in \mathcal{M}\}$ *and* $\Sigma_{in} = \{?a_{ij} \mid i, j \in 1..n \wedge a \in \mathcal{M}\}$. *The label* $!a_{ij}$ *denotes an output event that the instance Pi sends a message 'a' to Pj, and similarly an input event* $?a_{ij}$ *means Pj receives a message 'a' from Pi.*
- $msg : \Sigma_{out} \rightarrow \Sigma_{in}$ *is a bijection matching each output event to its corresponding input event, i.e.* $msg = \{x \in \mathcal{M}, i, j \in 1..n \bullet !x_{ij} \mapsto ?x_{ij}\}$.
- $\mathcal{O} : \mathbb{P}(\Sigma \times \Sigma)$ *is a relation on* Σ*, recording the constraint of general orderings in a bMSC. For a pair* $x \mapsto y : \Sigma \times \Sigma$, $x \mapsto y \in \mathcal{O}$ *if and only if there exists a general ordering symbol from* x *to* y.
- $<_C : \mathbb{P}(\Sigma \times \Sigma)$ *is a partial order on* Σ*. For each* $Pi \in \mathcal{P}$*, an adjacency relation* $<_i$ *denotes the top-to-bottom temporal ordering of the events occurring on Pi. The partial order* $<_C$ *is the transitive closure of the relation* $\ll_C$ *defined as*

$$\ll_C = \bigcup_{i:1..n} <_i \cup\, msg \cup \mathcal{O}.$$

The partial order $<_C$ is also known as the *causal ordering*, which can be understood as a visual order displayed in a bMSC diagram. The semantics of a bMSC specifically refers to its causal ordering. Unless specified otherwise, the partial orders in this paper are *strict* in the sense that they are *anti-reflexive*. Also note that the subscripts of the event labels, recording the origin and destination of a message, are of no importance here and therefore can be ignored. For a message $x \in \mathcal{M}$, its input and output events are represented as $?x$ and $!x$ respectively.

A collection of bMSCs can be composed together sequentially or conditionally. The High-level MSC, also known as hMSC or *road map*, is a structuring mechanism to compose bMSCs. In this paper, we only consider bMSCs, and the term MSC specifically refers to a bMSC.

2.2 Race Conditions

In discussions of operating systems or distributed systems, where at least two parallel processes have access to a single resource simultaneously, the term *race condition* refers to the situation where, without a synchronisation mechanism, inconsistencies may arise depending on which process wins the race to communicate with the resource. In the context of MSCs, however, a race condition does not match its usual meaning and therefore needs further explanation.

Initially discussed in [2], a race condition in an MSC refers to the likelihood that the implementation fails to obey the causal ordering described in its MSC specification. The concept can be illustrated via an example. $MSC3$ in Fig.2(a) shows a scenario with a race condition between the events $!b$ and $?c$. The specification requires that the input of c must follow the output of b. Nevertheless, $P3$ is specified to send out c after receiving a. Without querying $P2$, $P3$ has no knowledge of when $P2$ sends b. Therefore, in the implementation of such a protocol, it is quite possible that the message c arrives at $P2$ before $P2$ starts to send out b, which contradicts the specification.

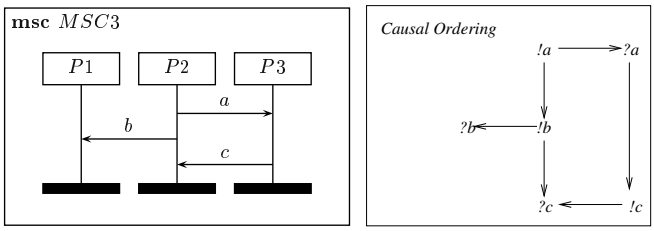

Fig. 2. (a)MSC with a race condition (b)Causal ordering $<_C$ of $MSC3$

Formal descriptions of a race condition can be found in the work of Alur et al. [2] and Mitchell [16] in different styles. In addition to causal ordering, Alur et al. [2] have defined two other levels of observation, i.e. *inferred ordering* and *enforced ordering*. In their method, detection of race conditions consists in checking if the inferred ordering is a subset of the transitive closure of the enforced ordering. They have also proved that detection of race conditions in a basic MSC[1] is decidable, and the tool uBET [11] from Bell Lab has been developed accordingly to address this problem.

Yet a solution which attempts to correct race conditions in an MSC had not emerged until Mitchell's work [16]. Here we quote directly the definition

[1] See [17] for detection of race conditions in High-level MSCs.

of a race condition in [16]. Intuitively, an MSC is race-free iff for an event x occurring before an input event $?e$, x must precede its corresponding output event $!e$ provided that x is not $!e$.

Definition 2. *An MSC is race-free when its causal ordering $<_C$ is race-free. A partial order $<$ on Σ is race-free if and only if*

$$x <?e \Rightarrow (x <!e \vee x =!e)$$

for every event $x \in \Sigma$ and message $e \in \mathcal{M}$.

In Mitchell's work [16], two solutions are proposed when given a causal ordering with race conditions. One is based on a race-free *inherent causal ordering* $<_I$, which is a minimal weakening of the causal ordering $<_C$. The other is the existence of a race-free *inherent refinement ordering* $<_R$, which is a minimal strengthening of $<_C$. The overall relationship is $<_I \subseteq <_C \subseteq <_R$. Both $<_I$ and $<_R$ have syntactically legal MSCs to match. The mapping between an ordering and its MSC format is trivial and can be automated. Here we are only concerned with $<_R$. The following definition differs from that in [16] to maintain $<_R$ a unique minimal strengthening of a causal ordering.[2]

Definition 3. *For a causal ordering $<_C$ on Σ, its inherent refinement ordering $<_R$ is the transitive closure of the relation $\ll_R$ defined as*

$$\ll_R = \ll_C \cup \{(x, !e) \in \Sigma \times \Sigma_{out} \mid x \ll_C ?e \text{ and } \neg (x \leq_C !e)\}.$$

Notations on partial orders. For a causal ordering $<_C$ on Σ and $x <_C y$ where $x, y \in \Sigma$, we say that x is an *immediate predecessor* of y if and only if $x \ll_C y$. For illustration, however, we use directed graphs to depict the relevant partial orders. Each vertex represents an element in Σ, and a directed edge is drawn from x to y, denoted by $x \rightarrow y$, whenever $x \ll_C y$. A path exists from x to y, denoted by $x \rightsquigarrow y$, iff $x <_C y$. Since $<_C$ is anti-reflexive, we have $x \not\rightsquigarrow x$, meaning a path consists of at least one edge in the graph. We let $\leq_C = <_C \cup Id(\Sigma)$ denote the reflexive version of $<_C$ such that $x \leq_C x$. The symbol $\rightsquigarrow_0$ is used to denote a path with zero or more edges, i.e. $x \rightsquigarrow_0 y$ iff $x \leq_C y$. The graph we use here is a variant of the *Hasse diagram* for a partial order in the sense that (1) the shape is different in order to maintain the similarity between a partial order and its MSC format, and (2) the edges are arrow-headed so that two events are ordered iff there is a path between them. For example, Fig.2(b) is the graph for the causal ordering $<_C$ of *MSC*3. It can be observed that the mapping between the directed graph and its MSC format is trivial. Also note that the variation pattern between $<_C$, $\ll_C$ and $\leq_C$ also applies to $<_R$.

The intuitive idea of constructing $<_R$ from $<_C$ of an MSC is to add ordering properties into $<_C$ so that the resulting $<_R$ satisfies Definition 2. Note that we

[2] Soundness problems of the original definition for the inherent refinement ordering in [16] will be discussed in our future exposition.

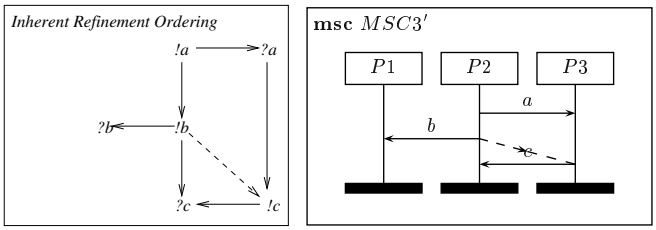

Fig. 3. (a)Inherent refinement ordering $<_R$ of $MSC3$ (b)Matching MSC scenario

use dashed edges for these additional constraints in a graph. Since the added ordering properties are facilitated by general ordering symbols in MSCs, dashed edges can maintain the resemblance between an inherent refinement ordering and its MSC format. As seen in Fig.2(b), $<_C$ of $MSC3$ has a race condition since $!b \rightsquigarrow ?c$ but $!b \not\rightsquigarrow !c$. The solution $<_R$ is constructed in Fig.3(a) by adding a dashed edge between $!b$ and $!c$, so that $<_R$ is race-free because $!b \rightsquigarrow ?c$ and $!b \rightsquigarrow !c$. The MSC scenario matching $<_R$ is $MSC3'$ in Fig.3(b).

3 Motivation

In the previous section, we have demonstrated that given an MSC scenario $MSC3$, its race-free counterpart $MSC3'$ can be derived by constructing $<_R$. A general ordering is used to delay events to avoid a race condition. In this case, $!c$ is delayed until $!b$ occurrs, so that $MSC3'$ is race-free. Nevertheless, further problems may arise when we consider how a general ordering can be implemented in a distributed environment. Mitchell [16] has indicated that the choice of implementation is up to the system designers who may use any mechanism that they deem *appropriate* for a particular circumstance. This effect can always be achieved by adding extra messages into the MSC with general orderings.

We recall the basic assumption of the MSC setting. Instances communicate asynchronously in a distributed environment without sharable resources. An instance can only get the information from others via message passing. Therefore, the above approach of adding messages is an effective way for specifiers to reveal explicitly how to implement the general orderings in an MSC specification. For example, in order to enforce the general ordering $!b \mapsto !c \in \mathcal{O}$ in $MSC3'$, an acknowledgement message can be added as shown in $MSC3''$ of Fig.4(a), where a silent symbol τ is used to label such a message since it does nothing but maintain the ordering. It can be easily noted that $MSC3''$ is not race-free since $?a <_C ?\tau_1$ but $?a \not<_C !\tau_1$, where $<_C$ is the causal ordering of $MSC3''$.

The problem continues even if we keep on building the inherent refinement ordering of $MSC3''$, which adds a general ordering between $?a$ and $!\tau_1$. Enforcing the general ordering with another message, say τ_2, gives us $MSC3'''$ as in Fig.4(b), where a race condition still exists between $!b$ and $?\tau_2$. It goes back to the case of $MSC3$. Without a more sophisticated approach, a simple MSC scenario can be flooded with unnecessary messages and becomes unreadable.

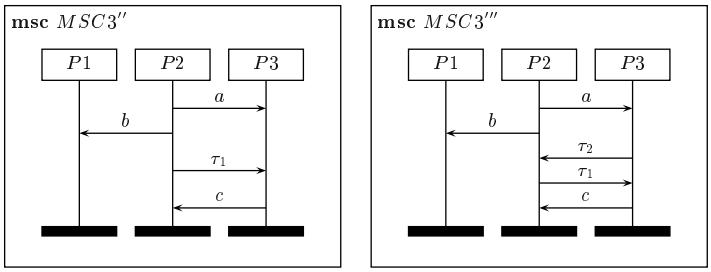

Fig. 4. (a)An acknowledgement message (b)MSC flooded with messages

In brief, our work is motivated by the observation that for a race-free inherent refinement ordering of an MSC with race conditions, an implementation that enforces the general orderings may not be race-free. The following definition explains the term *implementation* we use hereafter under the assumption that there is no general ordering symbols in the original MSC. Notationally, we use $Rel\, S$ as shorthand for the set of relations on S, i.e. $Rel\, S = \mathbb{P}(S \times S)$. We define a set A consisting of the events caused by the acknowledgement messages, i.e. $A = \{!\tau_i \mid i \in \mathbb{N}\} \cup \{?\tau_j \mid j \in \mathbb{N}\}$. The symbol $|S|$ denotes the number of elements in set S. For a relation R on set S, R^* is the transitive closure of R.

Definition 4. *For a causal ordering $<_C$ where $\mathcal{O} = \emptyset$ and its inherent refinement ordering $<_R$, the implementation of $<_R$ is a function $IMP : Rel\Sigma \rightarrow Rel(\Sigma \cup A)$ such that*

$$IMP(<_R) = (\Upsilon_1(\ll_R))^*$$

where the function $\Upsilon : Rel\Sigma \rightarrow Rel(\Sigma \cup A)$ is defined as follows. For a relation R on Σ, a message $x \in \mathcal{M}$, events $v_1, v_2 \in \Sigma$ and a counter $i \in 1..(|\ll_R \backslash \ll_C| + 1)$

$$\Upsilon_i(\ll_C) = \ll_C$$
$$\Upsilon_i(\{v_1 \mapsto !x\} \cup R) = \Upsilon_{i+1}(R) \cup \{!\tau_i \mapsto ?\tau_i, v_1 \mapsto !\tau_i, ?\tau_i \mapsto !x, v_2 \mapsto ?\tau_i\}$$

where $v_1 \mapsto !x \in \ll_R \backslash \ll_C$, $v_1 \mapsto !x \notin R$ and $v_2 \mapsto !x \in \ll_C$.

The intuition behind the above definition is to construct a causal ordering of the MSC in which an acknowledgement message is used to enforce a general ordering symbol. Here we use two graphs as an example. The relation $<_R$ depicted in Fig.5(a) shows the inherent refinement ordering of *MSC*3. A dashed edge denotes the difference between $\ll_C$ and $\ll_R$. The partial order $IMP(<_R)$ is displayed in Fig.5(b) by replacing the dashed edge $!b \mapsto !c$ in $<_R$ with $!\tau_1 \mapsto ?\tau_1$. Note that $IMP(<_R)$ is not race-free since $?a \rightsquigarrow ?\tau_1$ but $?a \not\rightsquigarrow !\tau_1$. The following proposition formalises this observation that motivates our work.

Proposition 1. $\neg$ ($\forall <_R$: $Rel\Sigma$, $<_R$ *is race-free* $\Rightarrow$ $IMP(<_R)$ *is race-free*).

This proposition can be easily justified by the counter-example we show in Fig.5. In the next section, we propose an approach to finding a partial order $<_{RF}$ that is stronger than $<_R$ such that both $<_{RF}$ and $IMP(<_{RF})$ are race-free.

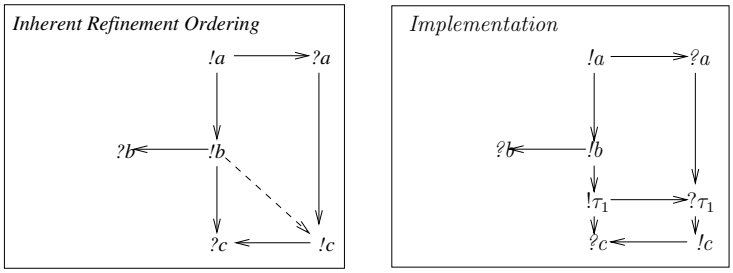

Fig. 5. (a)$<_R$ of $MSC3$ (b)IMP of $<_R$ of $MSC3$

4 Correcting Race Conditions

In this section, we propose our solution for correcting race conditions in an MSC scenario. The algorithm is expressed in a functional style with the aid of graphs explaining how the algorithm works. Soundness is then discussed in a more abstract way. Our intention is to make a formal argument to establish the correctness of our approach before coding it. Finally, a couple of simple examples are given for illustration.

4.1 Race-Free Refinement

We have observed that although an inherent refinement ordering is race-free, its implementation may not be. Our goal is to find another strengthened partial order $<_{RF}$, called *race-free refinement*, such that $<_R \subseteq <_{RF}$ and both $<_{RF}$ and $IMP(<_{RF})$ are race-free. We achieve this task by defining a function RF, which takes an inherent refinement ordering and non-deterministically returns a race-free refinement, i.e. $<_{RF} = RF(<_R)$. For simplicity, our approach is under the assumption that there is no general ordering construct in the original MSC diagram, which means all dashed edges appearing in the graphs are added by inherent refinement orderings.

The basic concept behind our approach can be understood via the two graphs in Fig.6. The symbol v stands for an event that can be either input or output. In Fig.6(a), the triangle $(v, !x, ?x)$ is a building block of all the inherent refinement orderings. The implementation of this graph will add a new input event, say $?\tau$, between $!y$ and $!x$. This addition causes a risk of a new race condition because $?\tau$ is a successor of $!y$, but $!y$ may not precede its corresponding output event $!\tau$, which is a successor of v. Nevertheless, no new race condition will occur if we have $?\tau$ precede $!y$, which means the dashed arrow should be lifted up to link v and $!y$ as shown in Fig.6(b). The new graph is still race-free because $v \rightarrow !y$ and $!y \rightarrow !x$ implies $v \rightsquigarrow !x$. This trivial example shows the most basic operation of the function RF, i.e. lifting up the dashed arrows in $<_R$ to a preceding output event. In a life-size MSC, however, there may be many output events preceding $!x$. In this case, we pick up the earliest one which does not precede v. To make this procedure more precise, we define the function ρ to perform this task.

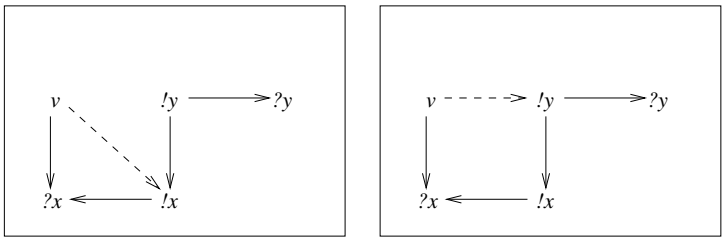

Fig. 6. (a)Graph for $<_R$ (b)Graph for $RF(<_R)$

Definition 5. *For a partial order $<$ on Σ and two events $u_1, u_2 \in \Sigma$, the function $\rho : Rel\Sigma \times \Sigma \times \Sigma \to \mathbb{P}(\Sigma_{out})$ is defined*

$$\rho(<, u_1, u_2) = min(\{v : \Sigma_{out} \mid v \leq u_2 \wedge v \not\leq u_1\})$$

where $min(S)$ returns a set of minimal elements of a partially ordered set S in the sense that an element a in S is called a minimal element if no other element of S strictly precedes a.

The function renders a set of minimal output events such that the events precede u_2 but do not precede or equal to u_1 with respect to the partial order $<$. The second parameter u_1 stands for the pivot-like event as v in Fig.6. The third parameter u_2 holds a place for the starting event, like $!x$, from which we trace back to a preceding output event. The application of ρ on the above example gives us $\rho(<_R, v, !x) = \{!y\}$ where $<_R$ represents the partial order depicted in Fig.6(a). Nevertheless, this example is too trivial in the sense that there is no event preceding $!y$. Simply replacing $v \mapsto !x$ with $v \mapsto !y$ gives us a race-free refinement. If we consider the case where some events precede $!y$, we need another mechanism to check whether the newly added edge $v \mapsto !y$ will cause a new race condition or not in the implementation. This mechanism can be explained more clearly after the function RF is defined.

Conventions on notation and designation are as follows. We let R, S range over $Rel\Sigma$. The lower-case letters e, r range over $\Sigma \times \Sigma$ and u, v over Σ. For a pair $e \in R$, $e.s \in \Sigma$ stands for the source vertex of the edge e and $e.d \in \Sigma$ for the destination vertex. We also define a replacement operator $[/]$ on a set S in the sense that $S[x/y] = \{x\} \cup S \backslash \{y\}$.

Definition 6. *For a causal ordering $<_C$ where $\mathcal{O} = \emptyset$ and its corresponding inherent refinement ordering $<_R$, the function $RF : Rel\Sigma \to Rel\Sigma$, which maps a partial order to a relation, is defined as*

$$RF(<_R) = (\Phi(\ll_R,\ \ll_R \backslash \ll_C))^*.$$

The function $\Phi : Rel\Sigma \times Rel\Sigma \to Rel\Sigma$ is defined inductively as

$$\Phi(R, \emptyset) = R$$
$$\Phi(R, \{e\} \cup S) = \begin{cases} \Phi(R, S) \backslash \{e\} & \text{if } e \in (\Phi(R, S) \backslash \{e\})^* \\ \Gamma(\Phi(R, S), r)[r/e] & \text{otherwise} \end{cases}$$

where $e \notin S$ *and*

$$r = e.s \mapsto v \text{ such that } v \in \rho((\Phi(R,S))^*, e.s, e.d).$$

Here we classify the right arrow $\rightarrow$ *into dashed or solid ones. A solid arrow, denoted by* $x \rightharpoonup y$*, describes the ordering formed by message arrows and instance axes. All other arrows are dashed, represented as* $x \rightharpoondown y$*, e.g. the edges in the set* $\ll_R \backslash \ll_C$ *or those generated by the functions* Φ *and* Γ*. Formally,* $\rightharpoonup = \ll_C$ *and* $\rightharpoondown = \rightarrow \backslash \rightharpoonup$ *where* $\rightarrow$ *refers to the binding occurrence of the relation* R *in* Γ*.*

The function $\Gamma : Rel\Sigma \times (\Sigma \times \Sigma) \rightarrow Rel\Sigma$*, which solves new race conditions that may arise when* r *is added by* Φ*, is defined as follows.*

$$\Gamma(R,r) = \begin{cases} R\backslash\{x \mapsto r.d\} & \text{if } \exists\, x : \Sigma_{out}\backslash\{r.s\}, x \rightharpoondown r.d \quad (1) \\ r_1 \cup \Gamma(R,r_1) & \text{if } \exists\, x : \Sigma_{in}\backslash\{r.s\}, x \rightharpoonup r.d \quad (2) \\ \Gamma(R,r_1)[r_1/x \mapsto r.d] & \text{if } \exists\, x : \Sigma_{in}\backslash\{r.s\}, x \rightharpoondown r.d \quad (3) \\ r_2 \cup \Gamma(\Gamma(R,r_1)[r_1/x \mapsto r.d], r_2) & \text{if } \exists\, x, y : \Sigma_{in}\backslash\{r.s\}, \\ & \qquad x \rightharpoondown r.d \wedge y \rightharpoonup r.d \quad (4) \\ r_1 \cup \Gamma(R,r_1)\backslash\{y \mapsto r.d\} & \text{if } \exists\, x : \Sigma_{in}\backslash\{r.s\}, y : \Sigma_{out}\backslash\{r.s\}, \\ & \qquad x \rightharpoonup r.d \wedge y \rightharpoondown r.d \quad (5) \\ R & \text{otherwise} \quad (6) \end{cases}$$

where

$$r_1 = x \mapsto v_1 \text{ such that } v_1 \in \rho(R^*, x, r.s)$$
$$r_2 = y \mapsto v_2 \text{ such that } v_2 \in \rho((\Gamma(R,r_1)[r_1/x \mapsto r.d])^*, y, r.s).$$

The algorithm for finding a race-free refinement consists of two recursive function calls, i.e. Φ and Γ. For each dashed edge in $<_R$, the basic operation of lifting that edge to a preceding output event up to its minimum is performed. This task is achieved by the iteration of the function Φ and can be illustrated in

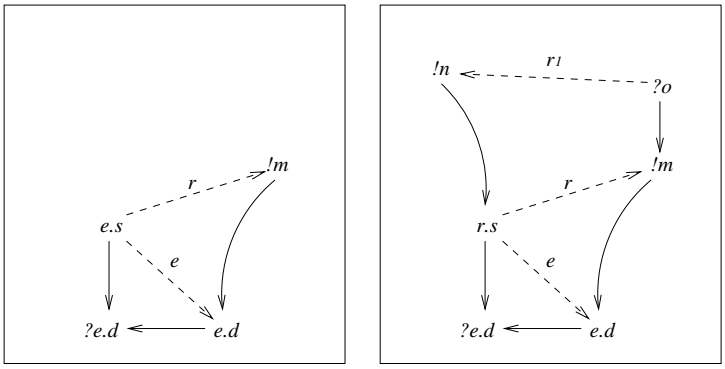

Fig. 7. (a)Behaviour of Φ (b)Behaviour of Γ

a more abstract way as shown in Fig.7(a). The curved edge from $!m$ to $e.d$ means $!m \rightsquigarrow_0 e.d$. The event $!m$ is the minimal element with the constraints such that $!m \rightsquigarrow_0 e.d$ and $!m \not\rightsquigarrow_0 e.s$. In the next section, we will prove in Lemma 1 that given a pair $e \in \ll_R \setminus \ll_C$, such an $!m$ always exists and could be simply $e.d$. The graph in Fig.7(a) depicts the behaviour of Φ, which replaces e with another pair $r = e.s \mapsto !m$. We can observe that an acknowledgement message enforcing r will add an input event preceding $!m$, causing no new race conditions.

The above description shows an ideal scenario where no event immediately precedes $!m$. Nevertheless, in some MSC scenarios, it is not unusual that there is another event immediately preceding $!m$, causing risks of new race conditions in the implementation. More precisely, if an input event, say $?o$, immediately precedes $!m$, the addition of r will *always* cause a new race condition due to $?o \not\rightsquigarrow r.s$, which we will prove later in Proposition 2. On the other hand, if an output event, say $!o$, immediately precedes $!m$, the addition of r will *never* cause a new race condition. We justify this feature here. From the definition of ρ, we can deduce $!o \rightsquigarrow_0 r.s$, otherwise $!o$ will be the $!m$. The acknowledgement message τ adds a pair $!\tau \mapsto ?\tau$ such that $r.s \rightarrow !\tau$ and $!o \rightarrow ?\tau \rightarrow !m$, which is still race-free since $!o \rightarrow ?\tau$ and $!o \rightsquigarrow_0 r.s \rightarrow !\tau$.

In Fig.7(b), we demonstrate the situation where an input event $?o$ immediately precedes $!m$. In this case, a new race condition will always occur in the implementation as mentioned earlier. This feature justifies the existence of the iteration of Γ within each recursive call of Φ. Tackling further race conditions caused by r is the task of Γ depending on how $!m$ is preceded. There are in total six cases expressed in the six equations in Γ. The graph of Fig.7(b) illustrates the equation (2) from Definition 6. The way we solve the problem is by finding another minimal output event, say $!n$, such that $!n \rightsquigarrow_0 r.s$ and $!n \not\rightsquigarrow_0 ?o$, and adding an edge $r_1 = ?o \mapsto !n$. Similarly, we can also prove that there always exists such an $!n$, which could be simply $r.s$. Note that $\Gamma(R, r_1)$ again appears at the right hand side of $=$ in the equation (2), which means checking if any new race condition may occur when r_1 is added. In this case, a new race condition occurs iff there exists at least one input event immediately preceding $!n$. The recursive structure of Γ basically forms an iteration to add or replace a set of edges $r_{1..n}$ after r is added under one recursive call of Φ. A more complicated structure of Γ is the equation (4), where there are two input events immediately preceding $!m$. The approach is similar to the above case except that the iteration of Γ occurs in one recursive call of Γ itself.

4.2 Soundness

Here we provide justification for the function RF that we claim can correct race conditions without introducing new ones when extra messages are added to enforce general orderings. Although RF is defined on partial orders, the style of our soundness discussion is based on analysing the behaviour of the function on the corresponding graphs, to which two template diagrams in Fig.7 serve as visual aids. The first proposition justifies the existence of Γ in the sense

that a new race condition will always occur in implementation if an input event immediately precedes the output event found by ρ in Φ, e.g. $?o \rightarrow !m$ in Fig.7(b).

Proposition 2. *In the context of Γ in RF, if there exists an event $x : \Sigma_{in} \backslash \{r.s\}$ such that $x \rightarrow r.d$, then $x \not\rightsquigarrow r.s$.*

Proof. There are two cases for $x \rightarrow r.d$. One is $x \rightharpoonup r.d$, and the other is $x \rightharpoondown r.d$. We show that $x \not\rightsquigarrow r.s$ in both cases.

Case 1: $x \rightharpoonup r.d$. Assume $x \rightsquigarrow r.s$; there are two possible routes, i.e. $x \rightharpoonup\rightsquigarrow_0 r.s$ and $x \rightharpoondown\rightsquigarrow_0 r.s$. The former case leads to a contradiction since $x \rightharpoonup r.d$ but $r.d \not\rightsquigarrow_0 r.s$. The latter case implies an event $v \in \Sigma$ such that $x \rightharpoondown v \rightsquigarrow_0 r.s$. Due to the nature of $\ll_R$, v must be an output event. So we have $x \rightharpoondown !y$ for an output event $!y$. The structure of $\ll_R$ enable us to deduce $x \rightharpoonup ?y$ from $x \rightharpoondown !y$, which contradicts the fact that $r.d$ is an output event.

Case 2: $x \rightharpoondown r.d$. This case implies $x \rightharpoonup ?r.d$, where $?r.d$ denotes the input event of $r.d$. Assuming $x \rightsquigarrow r.s$, there are two routes, i.e. $x \rightharpoondown\rightsquigarrow_0 r.s$ and $x \rightharpoonup\rightsquigarrow_0 r.s$. The former leads to a contradiction since $x \rightharpoondown r.d$ but $r.d \not\rightsquigarrow_0 r.s$. The latter also results in a contradiction because $x \rightharpoonup\rightsquigarrow_0 r.s$ implies $?r.d \rightsquigarrow_0 r.s$. Since $r.d \rightarrow ?r.d$ and $?r.d \rightsquigarrow_0 r.s$, we get $r.d \rightsquigarrow r.s$, which violates ρ. □

When explaining the mechanism of Φ, we have already mentioned that given a pair $e \in \ll_R \backslash \ll_C$, we can always find a minimal output event v such that $v \rightsquigarrow_0 e.d$ and $v \not\rightsquigarrow_0 e.s$. The same applies to Γ. Referring to Fig.7, we can say that $!m$ always exists in the application of Φ, and so does $!n$ in Γ. In the lemma below, we prove this property by showing that $!m$ could be simply $e.d$, and $!n$ could be simply either $r.s$ or the corresponding output event of $r.s$.

Lemma 1. *In the context of RF, the application of ρ gives a non-empty set.*

Proof. There are two places where the function ρ is called, i.e. Φ and Γ. In the case of Γ, the third parameter $r.s$ can be either an input or an output event.

Case 1: Φ. The event $e.d$ is a legitimate $!m$ because $e.d$ is an output event, and we also have $e.d \not\rightsquigarrow_0 e.s$ and $e.d \rightsquigarrow_0 e.d$.

Case 2: Γ. There are two sub-cases here.

Case 2.1: If $r.s$ is an output event, $r.s$ is a legitimate $!n$. We prove by contradiction. We assume $r.s$ is not a legitimate $!n$, which means $r.s \rightsquigarrow_0 ?o$. Since $?o \rightarrow !m$ and $!m \rightsquigarrow_0 e.d$, we can deduce $r.s \rightsquigarrow e.d$. In this case, the edge e would not have not existed in $<_R$ to correct a race condition in the first place, which is a contradiction.

Case 2.2: If $r.s$ is an input event, the corresponding output event of $r.s$, denoted by $!r.s$ is a legitimate $!n$. Similarly, we assume $!r.s$ is not a legitimate $!n$, i.e. $!r.s \rightsquigarrow_0 ?o$. We know $!r.s \neq ?o$, so $!r.s \rightsquigarrow ?o$. Since $<_R$ is race-free, $!r.s \rightsquigarrow ?o$ implies $!r.s \rightsquigarrow !o$. We also know that $!o \rightsquigarrow_0 r.s$, otherwise $!o$ will be the $!m$. Since $!o \neq r.s$, we have $!o \rightsquigarrow r.s$ hence $!o \rightsquigarrow !r.s$. A cycle arises in $<_R$ due to $!r.s \rightsquigarrow !o$ and $!o \rightsquigarrow !r.s$, which is a contradiction. □

We can observe that the mechanism of RF is to add or replace edges into the graph of an inherent refinement ordering $<_R$. The following lemma asserts that the edges added by RF are finite, which implies that the algorithm terminates. We prove the lemma by showing that each edge that may be added by one recursive call of Γ is between a pair of unordered input and output events.

Lemma 2. *For an inherent refinement ordering $<_R$, the application of RF inserts a finite number of edges into the graph of $<_R$.*

Proof. The structure of RF requires that for an edge e in $\ll_R \backslash \ll_C$, the function Φ replaces e with the edge r, and for each r, a set of edges $r_{1..n}$ may be added by Γ. Since $\ll_R \backslash \ll_C$ is finite, the number of edges replaced by Φ is finite. For an $r_i \in r_{1..n}$ added by Γ, there must exist an r_{i-1} and an input event, say $?o$, such that $?o \rightarrow r_{i-1}.d$ and $?o = r_i.s$. Proposition 2 asserts $?o \not\rightsquigarrow r_{i-1}.s$. Since $r_i.d \rightsquigarrow_0 r_{i-1}.s$, we have $?o \not\rightsquigarrow r_i.d$ before r_i is inserted. We also know that $r_i.d \not\rightsquigarrow_0 ?o$ from the definition of ρ. Since $r_i.d \neq ?o$, we have $r_i.d \not\rightsquigarrow ?o$. So an important feature of r_i arises in the sense that r_i only links a pair of unordered input and output events in $<_R$. We let the set of such pairs be U. We can assert $r_{1..n} \subseteq U \subseteq \Sigma \times \Sigma$. Since $\Sigma \times \Sigma$ is finite, the set $r_{1..n}$ is therefore finite. □

With the above lemma, we know that the task of RF consists in adding a finite number of edges into the graph of $<_R$. Here arises another question: how can we ensure that the relation $RF(<_R)$ is still a partial order? For an anti-reflexive relation, its transitive closure is a partial order if and only if there exists no cycle in the graph of that relation. Therefore our next observation is that the relation $RF(<_R)$ is a partial order by proving the graph of $RF(<_R)$ is acyclic in the following proposition.

Proposition 3. *For an $<_R$, the graph of $RF(<_R)$ is acyclic.*

Proof. Since the function RF recursively adds a finite number of edges, $r_{1..n}$, into the adjacency version of an inherent refinement ordering (Lemma 2), we can represent $RF(<_R)$ as $\ll_R \cup r_{1..n}$. We prove by induction on every $n \in \mathbb{N}$ that $RF(<_R) = \ll_R \cup r_{1..n}$ is acyclic.

Base case: Setting $n = 0$, we get $RF(<_R) = \ll_R$. Hence $RF(<_R)$ is cycle-free because $<_R$ is a partial order.

Induction steps: Let $n = k$ be an arbitrary number and suppose that $\ll_R \cup r_{1..k}$ is acyclic. When $n = k + 1$, we have $RF(<_R) = \ll_R \cup r_{1..k} \cup \{r_{k+1}\}$. There are two cases for r_{k+1}.

Case 1: r_{k+1} is generated by Φ, which means there exists an element $e \in \ll_R \backslash \ll_C$ such that $r_{k+1} = e.s \mapsto !m$ where $!m \in \rho((\ll_R \cup r_{1..k})^*, e.s, e.d)$. Due to Lemma 1, there will always be an $!m$ such that $!m \not\rightsquigarrow_0 e.s$, so r_{k+1} will not form a cycle.

Case 2: r_{k+1} is generated by Γ, which implies there exists an r_i in $r_{1..k}$ and an input event $?o$ such that $?o \rightarrow r_i.d$ and $r_{k+1} = ?o \mapsto !n$ where $!n \in \rho((\ll_R \cup r_{1..k})^*, ?o, r_i.s)$. Similarly, Lemma 1 asserts $!n$ always exists such that $!n \not\rightsquigarrow_0 ?o$. Hence r_{k+1} will not form a cycle.

Since r_{k+1} does not form a cycle in both cases, the graph of $RF(<_R)$ is acyclic. □

The final two propositions justify our intention in this paper. For an inherent refinement ordering $<_R$, we have found a strengthened partial order $<_{RF}= RF(<_R)$ such that both $<_{RF}$ and $IMP(<_{RF})$ are race-free. Note that the application of IMP on $<_{RF}$ requires its variant $\ll_{RF}$, which is actually the relation $\Phi(\ll_R,\ \ll_R \backslash \ll_C)$ before the transitive closure operator is performed.

Proposition 4. *For an $<_R$, $<_{RF}$ is race-free.*

Proof. We prove by contradiction. Supposing that $<_{RF}$ has race conditions. Since $<_R$ is race-free, for $<_{RF}$ to get a new race condition, the function RF must add an extra edge linking an event $v \in \Sigma$ with an input event $?x$ such that $v \rightarrow ?x$ but $v \not\rightsquigarrow !x$. This contradicts the definition of RF since all the edges added or replaced by RF link an event with an *output* event in the form of $v \rightarrow !x$. □

Proposition 5. *For an $<_R$, $IMP(<_{RF})$ is race-free.*

Proof. We prove by contradiction. Supposing that $IMP(<_{RF})$ is not race-free. The graph of the relation $<_{RF})$ must satisfy either one of the following two cases. We show that both cases contradict the mechanism of RF.

Case 1: $\exists\, r :\ll_{RF} \backslash \ll_C$ such that $v : \Sigma, v \neq r.s \wedge v \rightharpoonup r.d$.
In terms of the graph, this case holds when there are two dashed edges pointing to a single output event. This situation contradicts the definition of Γ no matter v is an input or an output event. If $v \in \Sigma_{out}$, the equation (1) or (5) of Γ applies, both of which eliminate the pair $v \mapsto r.d$ from the relation. If $v \in \Sigma_{in}$, the equation (3) or (4) is applicable, replacing $v \mapsto r.d$ with a different edge.

Case 2: $\exists\, r :\ll_{RF} \backslash \ll_C$ such that $v : \Sigma, v \neq r.s \wedge v \rightharpoonup r.d \wedge v \not\rightsquigarrow r.s$.
This case also contradicts the definition of Γ no matter v is an input or an output event. If $v \in \Sigma_{in}$, either equation (2) or (5) applies, and both equations add another edge so that $v \rightsquigarrow r.s$. If $v \in \Sigma_{out}$, then we have $v \rightsquigarrow r.s$. Otherwise, due to the definition of ρ, the edge r should have linked $r.s$ with v instead of $r.d$ in the first place. □

4.3 Examples

We use a couple of MSC scenarios with a reasonable level of complexity to illustrate our approach. For economy of space, we only show the MSC scenarios of an inherent refinement ordering and its corresponding race-free refinement instead of the partial orders. $MSC4$ in Fig.8 is an MSC with race conditions between three pairs of events, i.e. $(!a, ?c)$, $(!c, ?e)$ and $(?b, ?d)$. The MSC matching its inherent refinement ordering is depicted as $MSC4'$. In $MSC4'$, general ordering symbols are added to delay output events so that the resulting semantics satisfies the criteria of a race-free partial order. Nevertheless, if an acknowledgement message τ is used here to enforce the general ordering $!a \mapsto !c$, we can see that a new race condition arises between $!b$ and $?\tau$. This situation also applies to the other two general orderings $(!c, ?e)$ and $(?b, ?d)$.

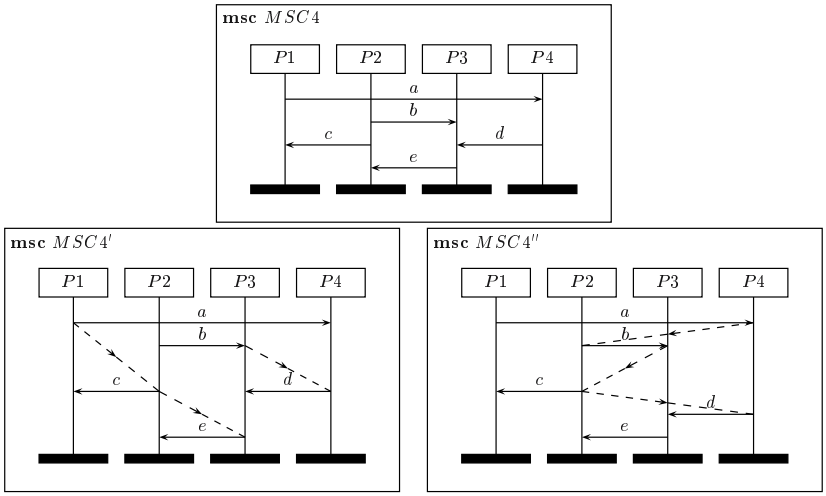

Fig. 8. A simple example

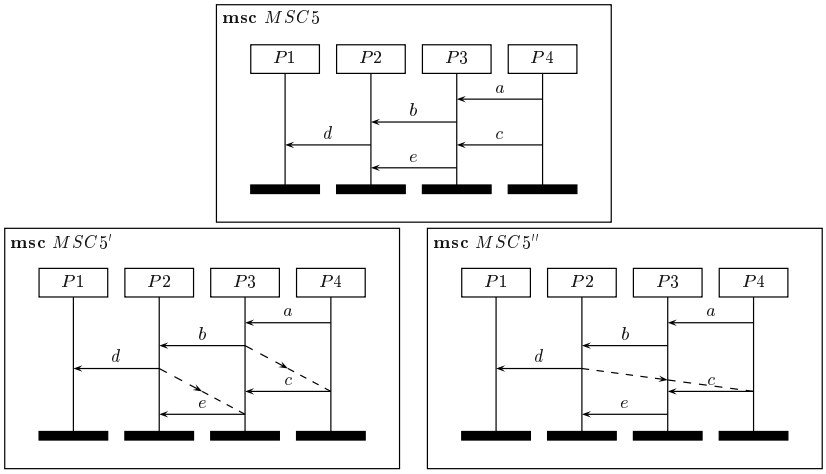

Fig. 9. A simple example

As illustrated beside $MSC4'$, $MSC4''$ in Fig.8 is the matching MSC scenario of the race-free refinement, i.e. $<_{RF}$, that we propose as the solution in this paper. In this case, adding acknowledgement messages to enforce the general orderings does not cause new race conditions.

The other example can be found in Fig.9, where $MSC5$ has race conditions in two places. Two general ordering symbols are added to form its inherent refinement scenario as shown in $MSC5'$. Our solution, however, requires only one general ordering symbol to solve the problem as depicted in $MSC5''$. As to the

implementation, $MSC5'$ needs two additional messages, causing new race conditions. Yet $MSC5''$ needs only one, ending up with another race-free scenario.

5 Conclusion

In this paper, we have investigated race conditions in a specification language, namely MSCs, based on partial-order semantics. The existing solution to correcting race conditions in an MSC is a canonical refinement, called inherent refinement ordering, which strengthens the causal ordering of the MSC with aid of general ordering constructs. We claim that new race conditions may occur if specifiers intend to reveal explicitly in MSCs how the general orderings are implemented by adding acknowledgement messages. This observation makes the inherent refinement orderings too weak for solving race conditions in practice.

This paper contributes an approach to finding a minimal strengthening partial order, called race-free refinement, of an inherent refinement ordering. We prove that for an inherent refinement ordering there exists a race-free refinement such that its matching MSC is race-free and remains race-free when acknowledgement messages enforcing general orderings are added. The approach is an algorithm on partial orders. The algorithm is presented in a functional style and can be later implemented for tool-support.

References

1. Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of Message Sequence Charts. *IEEE Transaction on Software Engineering*, 29:623–633, July 2003.
2. Rajeev Alur, Gerard J. Holzmann, and Doron Peled. An analyzer for Message Sequence Charts. *Software Concepts and Tools*, 17(2):70–77, 1996.
3. Hanêne Ben-Abdallah and Stefan Leue. Syntactic detection of process divergence and non-local choice in Message Sequence Charts. In *Proceedings of TACAS'97 (LNCS 1217)*, Netherland, April 1997. Springer-Verlag.
4. Grady Booch, Ivar Jacobson, and James Rumbaugh. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998. ISBN 0-201-30998-x.
5. Manfred Broy. On the meaning of Message Sequence Charts. In *Proceedings of the 1st Workshop of the SDL Forum Society Workshop on SDL and MSC*, volume I, pages 13–34, 1998.
6. CCITT. *CCITT Recommendation Z.120: Message Sequence Chart (MSC)*. Geneva, 1992.
7. C. Chen, S. Kalvala, and J. Sinclair. A process-based semantics for Message Sequence Charts with data. In *Australian Software Engineering Conference 2005 (ASWEC2005)*, Brisbane, March 2005.
8. W. Damm and D. Harel. LSCs: Breathing life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
9. Thomas Gehrke, Michaela Huhn, Arend Rensink, and Heike Wehrheim. An algebraic semantics for Message Sequence Charts documents. In *Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE/PSTV '98)*, Kluwer, 1998.

10. J. Grabowski, P. Graubmann, and E. Rudolph. Towards a Petri net based semantics definition for Message Sequence Charts. In *SDL'93 Using Objects*, Darmstadt, 1993. Proceeding of the 6th SDL Forum.
11. G. J. Holzmann, D. Peled, and M. H. Redberg. Design tools for requirements engineering. *Bell Lab Technical Journal*, 2(1):86–95, 1997.
12. ITU-TS. *Recommendation Z.120: Message Sequence Chart (MSC)*. Geneva, 1996.
13. P. B. Ladkin and S. Leue. Interpreting message flow graphs. *Formal Aspects of Computing*, 7(5):473–509, 1995.
14. P.B. Ladkin and S. Leue. What do Message Sequence Charts mean? In R.L. Tenney, P.D. Amer, and M.U. Uyar, editors, *Formal Description Techniques VI, IFIP Transactions C*, North-Holland, 1994. Proceeding of the 6th International Conference on Formal Description Techniques.
15. S. Mauw and M.A. Reniers. An algebraic semantics of basic Message Sequence Charts. *The Computer Journal*, 37(4):269–277, 1994.
16. Bill Mitchell. Inherent causal orderings of partial order scenarios. In *International Colloquim on Theoretical Aspects of Computing (LNCS 3407)*, China, September 2004. Springer-Verlag.
17. Anca Muscholl and Doron Peled. Message sequence graphs and decision problems on mazurkiewicz traces. In *MFCS*, pages 81–91, 1999.
18. A. Olsen, O. Færgemand, B Møller Pedersen, R. Reed, and J.R.W. Smith. *Systems Using SDL-92*. North Holland, 1994.
19. J. Schumann and J. Whittle. Generating statechart designs from scenarios. In *Proceeding of the 22nd International Conference on Software Engineering*, 2000.
20. S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Transaction on Software Engineering*, 29(2), February 2003.