# Controllability Problems in MSC-based Testing

Haitao Dan and Robert M. Hierons

*School of Information Systems, Computing & Mathematics*
*Brunel University, Uxbridge, Middlesex UB8 3PH, UK*
*Email: {haitao.dan, rob.hierons}@brunel.ac.uk*

**In testing systems with distributed interfaces/ports we may place a separate tester at each port. It is known that this approach can introduce *controllability problems* which have received much attention in testing from finite state machines. Message Sequence Charts (MSCs) form an alternative, commonly used, language for modelling distributed systems. However, controllability problems in testing from MSCs have not been thoroughly investigated. In this paper, controllability problems in MSC test cases are analysed with three notions of observability: local, tester and global. We identify two types of controllability problem in MSC-based testing. It transpires that each type of controllability problem is related to a type of MSC pathology. *Controllability problems of timing* are caused by races but not every race causes controllability problems; *controllability problems of choice* are caused by non-local choices and not every non-local choice causes controllability problems. We show that some controllability problems of timing are avoidable and some controllability problems of choices can be overcome when testers have better observational power. Algorithms are provided to tackle both types of controllability problems. Finally, we show how one can overcome controllability problems using a coordination service with status messages based on algorithms developed in this paper.**

*Keywords: Testing; Controllability Problems; Message Sequence Charts; Race; Non-local Choice*

*Received 00 January 2011; revised 00 Month 2011*

## 1. INTRODUCTION

Message Sequence Charts (MSCs) are a specification language suitable for describing the behaviour of distributed systems [1]. MSCs have become increasingly popular in the telecommunications and software industries and are widely used for requirements analysis, system design and formal verification [2, 3, 4, 5]. Sequence Diagrams (SDs), from the UML, are similar.

Model Based Testing (MBT) is a technique that automates the processes of test generation and execution on the basis of a model of the System Under Test (SUT). There has been much interest in MBT from both academia and industry [6, 7, 8, 9, 10, 11, 12, 13, 14, 15] and recent experience suggests that there can be significant resultant benefits [16]. However, there has been relatively little work on testing from MSCs [17, 18, 19, 20, 21]. This is in contrast to other popular behaviour models such as Finite State Machines (FSMs) and Input/Output Labelled Transition Systems (IOLTSs) [22, 23, 24, 25]. This seems surprising since there has been much interest in the use of MSCs and analysing MSCs to, for example, find pathologies [26, 4, 27, 28, 29, 30, 31, 32]. As MSCs are popular in modelling distributed systems, they have the potential to play an important role in MBT for distributed systems including systems based on grid and cloud technologies.

In our previous research [33], a formal conformance test framework was derived for testing from MSCs. Since MSCs model behaviours of multiple users and processes within the SUT, it leads to a complex test architecture in which there are multiple testers and system processes. Testers are used to simulate the users' behaviour in testing with an MSC architecture. In addition, testers may only have limited observability of the events that occur on system processes. Three different notions of observability, local, tester and global, were discussed [33]. When running MSC test cases with a limited observability, it is observed that testers may not be able to assure that the input to the SUT follows the given test cases. We say these are *controllability problems* in MSC-based testing. This problem has been investigated for other types of models with different test assumptions [34, 35, 36, 37, 38, 39, 40, 41].

The concept of controllability problems in distributed testing was first explored in the context of testing from a Deterministic Finite State Machine (DFSM) and in the situation in which the SUT interacts with its environment at two physically distributed interfaces, called ports [41, 36, 37]. It is assumed that the tester at a port $p$ only observes the input and output at $p$ and so cannot be aware of events at the other port. When testing from a DFSM $M$, a test sequence is a sequence $\sigma = x_1/y_1, \ldots, x_k/y_k$ of input/output pairs that is the label of a path of $M$ that starts at the initial state of $M$. In distributed testing, if there are $m > 1$ ports then an output $y_i$ is an $m$-tuple where the $p$th value of $y_i$ denotes the (possibly null) output sent to port $p$ in $y_i$. Let us suppose that there are two ports and we wish to apply a test sequence that involves the input of $x_1$ at port 1, this should lead to output $y_1$ at port 1 and then the tester at port 2 should supply input $x_2$. The problem here is that $x_2$ should be received by the SUT after the input/output pair $x_1/(y_1, -)$ but the tester at port 2 does not observe either the input or output from this pair and so cannot know when to send $x_2$. Thus, the testers cannot guarantee that the correct test is run and an apparent failure might be the result of a correct SUT receiving input in the wrong order.

When testing from a DFSM, a sequence $\sigma = x_1/y_1, \ldots, x_k/y_k$ of input/output pairs is considered to be *controllable* if for all $1 < i \leq k$, the tester to supply the input $x_i$ knows when to send this input. This is the case if and only if for all $1 < i \leq k$ we have that the tester to send $x_i$ either sent $x_{i-1}$ or observed an output in $y_{i-1}$. If this property does not hold for a test sequence $\sigma = x_1/y_1, \ldots, x_k/y_k$ then $\sigma$ is said to have a controllability problem. There has been significant interest in distributed testing from a DFSM and most work has either tried to avoid controllability problems [42, 35, 39, 43] or to add coordination messages between testers in order to overcome controllability problems [34, 44].

Most of the work on controllability problems in distributed testing has considered this in the context of testing from a DFSM. However, many distributed systems are non-deterministic and for such systems DFSM models are not suitable. While there has been some work on testing from a non-deterministic FSM [45] or an IOLTS [46], MSCs and SDs provide alternative but popular formalisms. Despite this, it appears that controllability problems have not previously been investigated for testing from such formalisms. When considering controllability problems in an MSC specification, the specification might either be a system model or it might model a proposed test scenario. For example, [17, 18] use MSCs as the behaviour model and generate test cases from them, but [19, 20] generally use MSCs to describe test purposes and test cases are generated in the form of TTCN (Tree and Tabular Combined Notation).

In this paper we analyse controllability problems in testing from MSCs under different types of observability. Except the discussions in Section 8, we assume that all the communications are asynchronous and non-FIFO. We give a definition of controllability problems based on whether MSC test cases lead to problematic test scenarios. In addition, two types of controllability problems are identified: controllability problems of timing and controllability problems of choice. Interestingly, we show that controllability problems of timing are related to the race pathology of MSCs [4] and controllability problems of choice are related to the non-local choice pathology [47]. Based on definitions of the two types of controllability problems, we build formal relationships between MSC pathologies and controllability problems. These relationships show that there is potential to adapt previous algorithms regarding MSC pathologies to deal with controllability problems in MSCs. Indeed, we give polynomial time algorithms (Algorithms 1 and 3) that capture controllability problems in testing from MSCs based on algorithms originally given in [4, 48, 31] for detecting race and non-local choice, respectively.

It is shown that some controllability problems of timing are avoidable by introducing an enforced order between a pair of observable events when testers have a better observational power than local observability. An algorithm (Algorithm 2) is given to find the races that lead to unavoidable controllability problems of timing under a given type of observability. It is also observed that some controllability problems of choice under local observability no longer exist when testers have better observabilities. We say that avoidable controllability problems of timing and reducible controllability problems of choice can be *overcome*. Consequently, a solution is proposed to overcome such controllability problems. We discuss the technique of implementing a coordination service with the support of status messages based on the knowledge of the controllability problems that can be overcome.

The remainder of this paper is organised as follows. In the next section, we introduce MSCs, the MSC test architecture and notions of observability for MSC testers. Based on the MSC test architecture, the definition of test case is derived to fit the multiple types of observability in Section 3. In Section 4, controllability problems in MSC test cases are analysed and defined. In addition, two subtypes of controllability problem are introduced. Section 5 analyses the relationship between controllability problems of timing and race. In Section 6, the relationship between controllability problems of choice and non-local choice is analysed. We then propose a solution to overcome controllability problems using a coordination service and status messages in Section 7. Section 8 provides a brief discussion on adapting the results in Section 5 and 6 when communications are FIFO. Finally, Section 9 presents conclusions and future work.

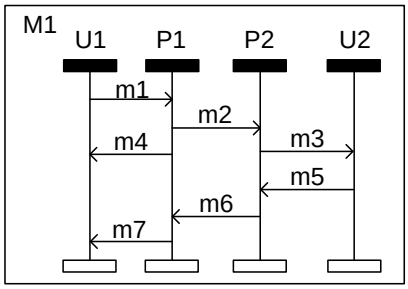


**FIGURE 1.** An MSC

## 2. PRELIMINARIES

In the first part of this section we briefly introduce MSCs and focus on the core constructs. We refer the reader to [1] for further information regarding MSCs. In the second part, we introduce the testing architecture and three possible types of observability in the MSC conformance test framework first given in [33].

### 2.1. MSCs

The graphical form of MSCs is straightforward as shown in Figure 1. A process in a distributed system is represented by a vertical line. Messages are horizontal or sloped lines exchanged between the processes and the direction of a message is denoted by the arrow at the end of the line. Events, usually only the sending and the receiving of messages, are represented by the end points of messages. Time progresses from top to bottom along the vertical lines [1].[1]

DEFINITION 2.1. (**MSCs**) *An* MSC $M$ *is a tuple* $\langle E, C, \mathcal{P}, l, msg, < \rangle$ *in which* $E$ *is a set of events,* $C$ *is the message alphabet and* $\mathcal{P} = \{P_1, \ldots, P_n\}$ *is a set of processes. The set* $E$ *is partitioned into a set* $S$ *of send events and a set* $R$ *of receive events* ($E = S \cup R$) *and* $l : E \mapsto \mathcal{A}$ *is a labelling function. We use* $send(i, j, m)$ *to represent the sending of message* $m$ *from* $P_i$ *to* $P_j$ *and* $receive(i, j, m)$ *the receiving of the corresponding message*[2]. *We define* $\mathcal{A} = \mathcal{A}^S \cup \mathcal{A}^R$ *where* $\mathcal{A}^S = \{send(i, j, m) : \ 1 \leq i, j \leq n \ \wedge \ m \in C\}$ *and* $\mathcal{A}^R = \{receive(i, j, m) : \ 1 \leq i, j \leq n \ \wedge \ m \in C\}$. *We use* $\mathcal{A}_i$ *to represent the set of labels on process* $P_i$ *and* $msg : S \mapsto R$ *is a bijection from send to receive events, matching each send with its corresponding receive, the inverse mapping being* $msg^{-1} : R \mapsto S$. *We also use a helper mapping* $p : E \mapsto [1, n]$ *that maps each event* $e \in E$ *to the index of the process on which* $e$ *occurs. For each* $1 \leq i \leq n$, *there is a total order* $<_i$ *on the events of process* $P_i$ *such that the transitive closure of the relation* $< \doteq \bigcup_{1 \leq i \leq n} <_i \cup \{(s, msg(s)) : s \in S\}$ *is a partial order on* $E$, *namely the* visual order ($<^*$).

**Example** (**Event, label and visual order of an MSC**) Consider MSC $M1$ in Figure 1.[3] The sending of $m1$ is an event $e$ on process $U1$ and this has label $send(U1, P1, m1)$ in $\mathcal{A}$. There is an event $e'$ with label $receive(U1, P1, m1)$ such that $msg(e) = e'$ and $e <^* e'$. The event of receiving of $m7$ on $U1$ is preceded under $<^*$ by the sending of $m1$ and the receiving of $m4$. □

Throughout this paper, we discuss MSCs under the non-degeneracy condition. Non-degeneracy means that degenerate MSCs are not allowed. An MSC is degenerate if two send events $e1$ and $e2$ exist such that $l(e1) = l(e2)$, $e1 < e2$ and $msg(e2) < msg(e1)$. Based on the non-degeneracy condition, a word $w = w_1 \cdots w_{|E|}$ over the alphabet $\mathcal{A}$ is a *word* of an MSC $M$ if there exists a total order $e_1 \cdots e_{|E|}$ of the events in $E$ such that whenever $e_i < e_j$ we have $i < j$, and for $1 \leq i \leq |E|$, $w_i = l(e_i)$. The word is *well-formed* if for each receive event there is a corresponding send event and is *complete* if all send events have matching receives. For an MSC $M$, the language of $M$, $L(M)$, is the set of all words of $M$ [48].

In this paper, we consider MSC specifications that contain a finite number of MSCs.

DEFINITION 2.2. (**MSC Specification**) *An MSC specification,* $\mathcal{M}$ *is a finite set of* $k$ *MSCs* $M_1, \ldots, M_k$ *in which* $M_j$ *has event set* $E_j$, *message alphabet* $C_j$ *and set* $\mathcal{P}_j$ *of processes and* $E_1, \cdots, E_k$ *are disjoint.* $\mathcal{E} = \bigcup_{j=1}^{k} E_j$ *is the sets of events of* $\mathcal{M}$; $\chi = \bigcup_{j=1}^{k} C_j$ *is the message alphabet of* $\mathcal{M}$; $\psi = \bigcup_{j=1}^{k} \mathcal{P}_j$ *is the set of processes in* $\mathcal{M}$.

The language of an MSC specification is the union of the languages of all MSCs in $\mathcal{M}$, $L(\mathcal{M}) = \bigcup_{j=1}^{k} L(M_j)$. We note that the message alphabets and process names of different MSCs in an MSC specification need not be disjoint. From an MSC specification $\mathcal{M}$, the process behaviour of $P_i$ can be derived from $L(\mathcal{M})$, namely the process language.

DEFINITION 2.3. (**Process Language**) *The process language at* $P_i$ *of an MSC specification* $\mathcal{M}$ *is the projection of* $L(\mathcal{M})$ *onto* $P_i$, *denoted* $L(\mathcal{M})|\mathcal{A}_i$.

In Definition 2.3, | is used as the projection operator which only keeps letters in alphabet $\mathcal{A}_i$ and removes others from the words of the language. For an MSC specification, it is clear that there may be multiple possible behaviours that can be chosen from after a common prefix for a process. Let *pref*($L$) represent the set of prefixes of words from language $L$, possible behaviours form *choices* which are defined as follows.

DEFINITION 2.4. (**Choice** [31]) *Given an MSC specification* $\mathcal{M}$ *and process* $P_i$, *a* choice *on* $P_i$ *is a*

[1] Coregion is not considered in this paper. However, the main results of this paper are capable of dealing with MSCs with coregions, because an MSC with coregions can be transformed to a set of MSCs without coregions.

[2] We will use $!m$ and $?m$ as abbreviations of $send(i, j, m)$ and $receive(i, j, m)$, respectively, where $i, j$ are clear.

[3] In this paper, $U*$ and $P*$ may be used to represent user and system processes in figures, respectively.

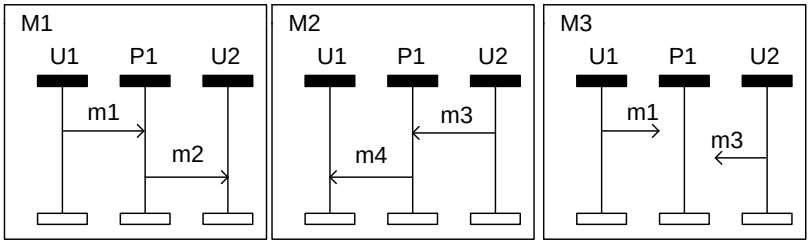


**FIGURE 2.** An example of controllability problems of choice

*triple* $(w, x, y)$*, where* $w \in pref(L(\mathcal{M})|\mathcal{A}_i)$*,* $x, y \in \mathcal{A}_i$ *and* $x \neq y$ *such that* $wx, wy \in pref(L(\mathcal{M})|\mathcal{A}_i)$*.*

**Example** (**Choice**) Consider MSC specification $\mathcal{M}1$ formed by $M1$ and $M2$ shown in Figure 2. There is a choice, $(\varepsilon, !m1, ?m4)$, on process $U1$. Here, $\varepsilon$ represents the empty word. This choice on $U1$ means that, from the start, $U1$ can choose to send $m1$ or wait for incoming $m4$. □

### 2.2. Testing from MSCs

Conformance testing is about checking whether the behaviour of a system conforms to the corresponding specification. An MSC specification generally involves multiple user and system processes. For testing from MSC specifications, testers thus simulate the users' behaviour according to the specification and check the conformance by comparing the observation of the SUT to the system behaviour described by the specification. In MSCs, a system process may communicate with many users and other system processes through channels set up between each pair of these entities. Therefore, the SUT described by the MSC specification contains multiple subsystems, each subsystem has multiple ports and a tester may communicate with any of the ports.

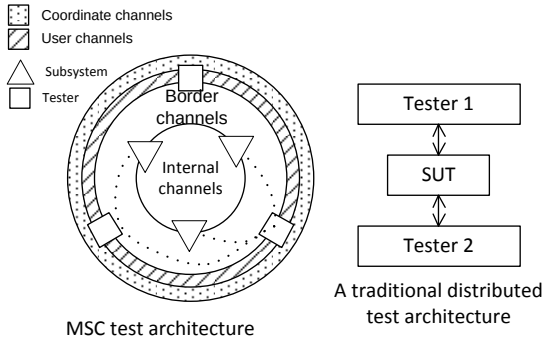


**FIGURE 3.** Distributed test architectures

The traditional distributed test architecture is shown on the right-hand side of Figure 3 [38, 39, 40] in which the SUT has two distributed ports each of which communicates with only one tester. It is clear that the SUT described by an MSC specification requires a more complex test environment. Therefore the *MSC test architecture*, shown on the left-hand side of Figure 3, was developed [33]. In the MSC test architecture, triangles are used to represent system processes and squares are used to represent testers. Interactions between entities are transmitted through communications channels. The communications channels are divided into three groups: border, user and internal channels which correspond to channels between one user process and one system process, two user processes and two system processes, respectively. The groups of channels are represented by concentric circles in the test architecture given in Figure 3. The inner circle denotes the internal channels; the first ring out from the inner circle denotes border channels; the second ring denotes the user channels. Real channels may be set up for pairs of entities. In addition to these three types of channels, there may be coordination channels between the testers and these allow them to exchange coordination messages. In Figure 3, coordination channels are represented by the outer ring in the architecture. These differ from user channels since messages exchanged in coordination channels are not from the MSCs; they are included in order to assist the test execution. When there are distributed testers, the observational power of a single tester depends upon the testing infrastructure used. For example, the separate testers may be able to communication through coordination channels and testing might use a specific technology such as monitoring systems [49]. In this paper, we discuss controllability problems based on three notions of observability.

- Local observability: a tester can only observe events on its process due to the distributed testing architecture.
- Tester observability: in addition to the events on itself, one tester shares the information with other testers by communication through coordination channels.
- Global observability: testers can observe all events including those on processes within the SUT.

## 3. MSC TEST CASES

Test cases can be derived from an MSC specification to check whether the behaviour of the SUT conforms to the specification. Individual testers in an MSC test

architecture obtain verdicts, such as pass and fail, by comparing their observations to the specification.

As mentioned in Section 2, testers in the MSC test architecture may have different types of observability. For a tester, the observation of a test run varies with different types of observability.

**Example** (**Types of observability**) Consider a test run following MSC $M1$ given in Figure 1, with testers $T1$ and $T2$ simulating user processes $U1$ and $U2$, respectively. With local observability, $T2$ can observe events $?m3$ and $!m5$. In addition to $?m3$ and $!m5$, $T2$ can also observe $!m1$ and $?m4$ and $?m7$ with tester observability since testers share information. If $T2$ has global observability, all events in $M1$ can be observed by $T2$. □

Consequently, the set of possible valid observable sequences of the SUT changes with the type of observability; the word that captures an execution of the SUT may conform to an element of $L(\mathcal{M})$ under one notion of observability but not another. For MSC specification $\mathcal{M}$ with a tester simulating process $P_i$, $L(\mathcal{M})$ is the set of the desirable observations of a tester with global observability. If the tester has local observability, a word of process language $L(\mathcal{M})|\mathcal{A}_i$ describes a valid observation. Let $P_1, .., P_k$ be the set of user processes and $\mathcal{A}_u = \bigcup_1^k \mathcal{A}_i$ be the alphabet on user processes. With tester observability, the set of valid observable sequences is $L(\mathcal{M})|\mathcal{A}_u$. Obviously, global observability gives the testers the greatest ability to distinguish between the behaviours of the SUT and the specification.

To give a unified definition, a test case for MSC-based testing is defined based on the possible observable sequences from each participating tester. The observation of every tester in a test run should conform to the MSC specification and otherwise the SUT fails the test. Let $\mathcal{A}^i$ denote the observable alphabet of the tester simulating user process $P_i$. Alphabet $\mathcal{A}^i$ changes with the type of observability. For example, $\mathcal{A}^i = \mathcal{A}_i$ with local observability; $\mathcal{A}^i = \mathcal{A}_u$ with tester observability and $\mathcal{A}^i = \mathcal{A}$ with global observability. The set of observable sequences for the tester under a type of observability can be formalised as the projection of the MSC language on $\mathcal{A}^i$ and we call this the *tester language*. We denote the tester language of user process $P_i$ as $L_i = L(\mathcal{M})|A^i$. A tester language $L_i$ is the process language if we have local observability; $L_i$ is $L(\mathcal{M})|\mathcal{A}_u$ with tester observability and $L_i$ is $L(\mathcal{M})$ with global observability. The notion of a test case can be formally defined as follows.

DEFINITION 3.1. (**MSC Test Case**) *Given an MSC specification $\mathcal{M}$ with $k$ user processes $\{P_1, ..., P_k\}$, $1 \leq i \leq k$, an MSC test case of $\mathcal{M}$ is a group of tester languages $\mathcal{T} = (L_1, ..., L_k)$.* [4]

[4]For both tester and global observabilities, $L_1 = L_2 = ... = L_k$. In these cases, the definition can be simplified. However, the given definition has a unified form and supports possible fine-grained types of observabilities in complex distributed testing environments.

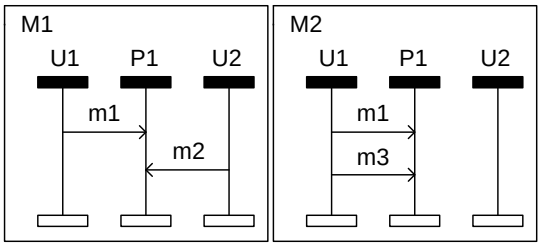


**FIGURE 4.** Controllability problems of timing

This definition says that, in executing a test case, each tester follows its tester language. This means that only 'sensible' positive (send) events can happen. From the other perspective, this also means that a tester can make a decision on a choice according to the tester language, simply because any decision on a choice conforms to the tester language.

The verdicts of test runs are given by all testers. Each tester gives pass verdicts if its observations conform to its tester language, otherwise fail verdicts are given.

## 4. CONTROLLABILITY PROBLEMS

According to the definition of an MSC test case, testers can provide the input to and wait for the output from the SUT following the test specification (tester language). One question is whether this is enough to assure that all testers behave properly and provide desirable input to the SUT. We will see that a tester might not have enough information to ensure that it makes correct decisions.

The time between behaviours of a tester is not normally defined in an MSC specification. So, in a test run, if a tester on process $P_i$ is to supply an input then it might do so immediately or choose a delay. We will see that the overall behaviour of the test run can depend on the delay. In addition a tester may confront choices (Definition 2.4). Some options of choices may lead to problematic test scenarios.

The first problem is that, due to the absence of the information on transmission time in MSCs, there are cases where messages from the testers arrive at the SUT in a wrong order. As a result, even if all testers follow their tester languages, an observed failure may not be caused by implementation problems of the SUT but by an error in the order in which messages from testers are received.

**Example** (**Problem of timing**) Consider the test case for the MSC $M1$ given in Figure 4. Here two testers $U1$ and $U2$ should send messages $m1$ and $m2$ respectively to process $P1$ of the SUT. According to the specification $M1$, message $m1$ should arrive at $P1$ before message $m2$. With local or tester observability,

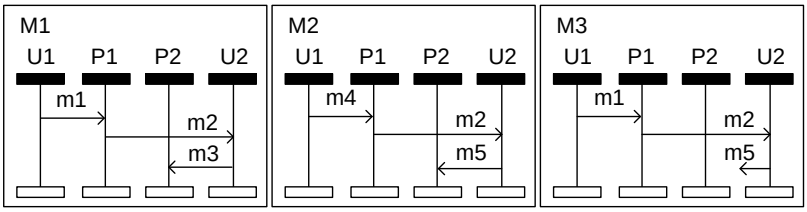


**FIGURE 5.** Another example of controllability problems of choice

$U2$ cannot know when $P1$ receives $m1$. Therefore, it is possible that the testers send their messages as desired but $m2$ arrives at $P1$ before $m1$ in a test run. Another example is given in $M2$ shown in the same figure. In this example, two messages are sent from tester $U1$. As described in $M2$, $m1$ should arrive before $m3$. The problems is that the tester simulating $U1$ cannot assure that $m3$ always arrives after $m1$, because $m3$ may overtake $m1$ in a test run. □

Another type of problem can occur when a tester $T_i$ has made a choice on a send event according to its tester language, but this choice is forbidden as a result of a previous decision made by another process $P_j$. The problem is that the tester $T_i$ cannot be aware of the choice made by $P_j$. In such a situation, a decision by $T_i$ regarding a choice that involves the sending of a message may lead to test scenarios that violate the MSC specification.

**Example** (**Problem of choice**) Consider MSC specification $\mathcal{M}1$ containing MSCs $M1$ and $M2$ shown in Figure 2, the valid test runs described by $\mathcal{M}1$ are $!m1?m2$ and $!m3?m4$. Let us suppose that the tester simulating $U1$ chooses to send $m1$. The choice between sending of $m3$ and receiving of $m2$ is non-deterministic for $U2$ under local observability since the tester simulating $U2$ cannot know what has been sent from $U1$. $M3$ thus can be the actual run of the test case and $M3$ is obviously an undesirable test scenario. Another example is given in Figure 5 that shows the MSC specification $\mathcal{M}2$ with two MSCs $M1$ and $M2$. The non-deterministic choice happens after $U2$ receives $m2$ with local observability. $U2$ cannot decide whether to send $m3$ or $m5$ since $U2$ does not know what has been sent from $U1$ to $P1$. Therefore, $M3$ can be the test run which is undesirable. □

Base on these observations, we define the problematic test scenarios as follows.

DEFINITION 4.1. (**Problematic Test Scenario**) *Given an MSC specification $\mathcal{M}$ with alphabet $\mathcal{A}$, $wa$, where $a = l(f)$, is a problematic test scenario if $wa$ is a well-formed word such that $w \in pref(L(\mathcal{M}))$ and:*

- (Type 1) *$f \in R$ is on a system process $P_i$ and is sent from a user process and $wa|\mathcal{A}_i \notin pref(L(\mathcal{M})|\mathcal{A}_i)$;*

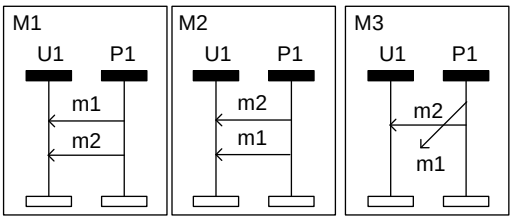


**FIGURE 6.** A non-local choice causes no controllability problems

- *or* (Type 2) *$f \in S$, $f$ is on a user process $P_j$, $wa|\mathcal{A}^j \in pref(L^j)$ and $wa \notin pref(L(\mathcal{M}))$.*

**Example** (**Type 1 problematic scenario**) In the MSC $M1$ given in Figure 4, two testers $U1$ and $U2$ should send messages $m1$ and $m2$ respectively. However, $U2$ cannot know when to send its message in order to ensure that $P1$ receives $m2$ after $m1$. Based on Definition 4.1, $!m1!m2?m2$ can be the problematic test scenario and we have $w = !m1!m2$ and $a = ?m2$. □

**Example** (**Type 2 problematic scenario**) Let the testers have local observability in Figure 5, where $U1$ and $U2$ have choices in MSC specification $\mathcal{M}2$ that contains $M1$ and $M2$. The problem is that $U2$ has to decide whether to send $m3$ or $m5$ after receiving $m2$. In this case, we have two combinations of $w$ and $a$ that satisfy Definition 4.1: one is $w = !m1?m1!m2?m2$ and $a = !m5$; the other is $w = !m4?m4!m2?m2$ and $a = !m3$. □

It is required that $f \in S$ (sending of a message) in Type 2 problematic test scenarios. Receive events are ruled out based on a common assumption in research on formal conformance testing: testers cannot block output from the SUT [46]. The undesired incoming messages to testers may lead to scenarios violating the MSC specification, but the resultant scenarios are not controllable for testers. Therefore, they are excluded from problematic test scenarios.

**Example** (**Explanation of the sending of message requirement**) Consider MSC specification $\mathcal{M}3$ with two MSCs, $M1$ and $M2$ shown in Figure 6, in which there is a possibility that system process $P1$ decides to send $m1$ then $m2$ but $m2$ overtakes $m1$ and arrives at user process $U1$ first. This scenario is shown in $M3$ in the same figure. The tester simulating $U1$ cannot

decide that $M3$ is a problematic run since the receiving of $m2$ is described by $M2$. However, $M3$ already violates the specification since $M3$ is not described by the specification. Obviously, $M3$ is not controllable for the tester in this case. □

The relationship between the two types of problematic test scenario are stated in the following proposition.

PROPOSITION 4.1. *Type 1 and Type 2 problematic test scenarios are disjoint.*

*Proof.* This follows immediately from the event $f$, where $a = l(f)$, in the problematic scenario $wa$ being a receive event in a Type 1 scenario and a send event in a Type 2 scenario. □

Controllability problems of MSC test cases lead to problematic test scenarios and are results of design problems in MSC specifications.

DEFINITION 4.2. (**Controllability Problems of MSC Test Cases**) *Given an MSC specification $\mathcal{M}$, its MSC test case $\mathcal{T}$ has controllability problems if there can be problematic test scenarios even if each tester follows its tester language.*

Both designers and testers can benefit from techniques that detect and avoid controllability problems. For system designers, it is desirable to avoid specific types of controllability problems and this might be achieved by detecting the corresponding types of design problems at an early stage. Testers can benefit explicitly when parts of designs are being considered as potential test cases. Interestingly, in the following sections we show that controllability problems in MSC test cases are highly related to two types of MSC pathologies: race and non-local choice [4, 28, 31]. Therefore, we investigate the relationships between controllability problems and the existing research on MSC pathologies [50, 26].

## 5. CONTROLLABILITY PROBLEMS OF TIMING

In this section, we first give definitions of controllability problems of timing and of race in an MSC and in an MSC specification. Based on these definitions, we derive the relationship between race and controllability problems of timing. It is also shown that some controllability problems caused by races can be avoided by introducing enforced orders between observable events. Finally, algorithms are provided to check whether a test case of an MSC specification has controllability problems of timing and whether the controllability problems can be overcome.

As mentioned in Section 4, controllability problems can be classified by the types of problematic test scenarios that they cause. The Type 1 problematic test scenarios are caused by the testers not knowing when to send input. Therefore, we call them controllability problems of timing.

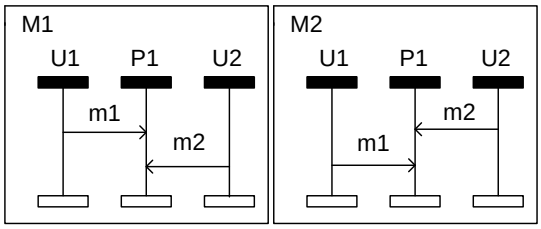


**FIGURE 7.** Races in an MSC specification

DEFINITION 5.1. (**Controllability Problems of Timing**) *Given an MSC specification $\mathcal{M}$, its MSC test case $\mathcal{T}$ has controllability problems of timing if testers may lead to Type 1 problematic test scenarios.*

### 5.1. Race

Race in an MSC defines discrepancies between two orders: the visual order and the enforced order. This means that the visual order between some events cannot be assured in executions. The enforced order depends on the underlying communication system [4]. In this paper, since we assume that communications between processes are asynchronous and non-FIFO, enforced order among the events in MSCs is the same as that produced when making no assumptions on communication systems. In both situations, causalities are the only constraints on the orders among the events in MSCs. The causalities are defined by the sending of a message always being before the receiving of the same message and a send event on some process always being after the events visually above it on the same process. So the enforced order of an MSC is defined as follows.

DEFINITION 5.2. (**Enforced Order**) *Given an MSC $M$ with set $E$ of events, the transitive closure of the relation, $\ll \doteq \{(x, y) \in <: y = msg(x) \vee (y \in S \wedge p(x) = p(y))\}$, is a partial order on $E$, namely the* enforced order ($\ll^*$).

Races in an MSC are defined as follows [4].

DEFINITION 5.3. (**Race in an MSC**) *Events $e$ and $f$ from process $P_i$ in MSC $M$ are said to be in a* race, *denoted as $[e, f]$, if $e <^* f$ but not $e \ll^* f$.*

However, races in an MSC specification are not the union of the races in each member MSC of the MSC specification.

**Example** (**Race in an MSC specification**) Consider MSC specification $\mathcal{M}4$ with two member MSC $M1$ and $M2$ shown in Figure 7. In each MSC, the event pair, $?m1$ and $?m2$, forms a race on process $P1$, but no matter whether $m2$ arrives before $m1$ or the converse situation happens, they are both allowed by the specification $\mathcal{M}4$. □

Therefore, a race in an MSC specification should first be a race of a member MSC and, in addition the problematic scenario caused by the race should not be described by the specification. The definition of race in MSC specifications is as follows and was first given in [32].

DEFINITION 5.4. (**Race in MSC Specifications**) *Let us assume that $\mathcal{M}$ is an MSC specification and $e, f$ are two events on process $P_i$. $[e, f]$ is a* race *of $\mathcal{M}$ if $[e, f]$ is a race in some member MSC $M$ of $\mathcal{M}$ such that $uau'b \in pref(L(M)|\mathcal{A}_i)$ where $a = l(e)$, $b = l(f)$ and $u, u'$ are two words in alphabet $\mathcal{A}_i$ and $ub \notin pref(L(\mathcal{M})|\mathcal{A}_i)$.*

In the definition, $uau'b$ represents a prefix of the projection of $M$ on process $P_i$. It contains the labels of the two events which form race $[e, f]$. $ub$ is the projection of a partial scenario $M'$ in which $f$ overtakes $e$. $M'$ is therefore the problematic scenario caused by the race $[e, f]$.

### 5.2. Race and controllability problems

Race causes controllability problems of timing.

**Example** (**Controllability problem and race**) Let us reconsider the examples given in Figure 4. The controllability problems in $M1$ and $M2$ are actually caused by races in those MSCs. $[?m1, ?m2]$ is a race in $M1$ since $?m1 <^* ?m2$ but $?m1 \not<^* ?m2$ and $[?m1, ?m3]$ is a race in $M2$ since $?m1 <^* ?m3$ but $?m1 \not<^* ?m3$. □

Not every race in MSCs causes controllability problems. A race will not cause a controllability problem if the second event of the race is a system process receiving a message that is sent from a system process.

**Example** (**Race not causing controllability problems 1**) $[?m2, ?m3]$ is a race of MSC $M1$ in Figure 8 which may lead to a new scenario in which $m3$ arrives before $m2$. The actions of the testers does not influence this and the SUT may contain mechanisms that ensure that such a scenario cannot occur. □

In addition, if a race is on a user process, it will not cause controllability problems but may cause a fail verdict.

**Example** (**Race not causing controllability problems 2**) $[?m4, ?m5]$ is a race of MSC $M2$ in Figure 8 which may lead to a scenario in which $m5$ overtakes $m4$. This undesired behaviour is observable for tester $U1$. If this occurs then $U1$ will give a failure verdict to the test run since the observations are not consistent with the specification. □

We can conclude the relationship between races and controllability problems of timing as follows.

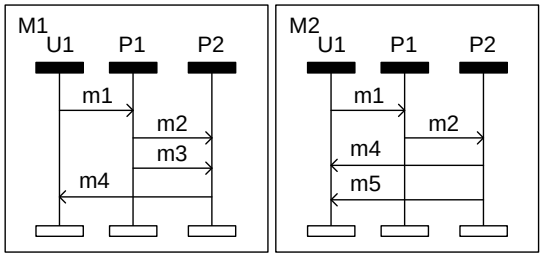


**FIGURE 8.** Races but no controllability problems

PROPOSITION 5.1. *A test case $\mathcal{T}$ of the MSC specification $\mathcal{M}$ has controllability problems of timing if and only if there is a race $[e, f]$ of $\mathcal{M}$ on a system process $P_i$ such that $msg^{-1}(f)$ is on a user process.*

*Proof.* First let us suppose that the test case $\mathcal{T}$ of $\mathcal{M}$ causes no Type 1 problematic test scenarios; we then prove that there is no race $[e, f]$ in $\mathcal{M}$ on a system process $P_i$ such that $msg^{-1}(f)$ is on a user process. We will use proof by contradiction. Let us assume that $\mathcal{M}$ causes no Type 1 problematic test scenarios but there is a race $[e, f]$ of $\mathcal{M}$ on a system process $P_i$ such that $msg^{-1}(f)$ is on a user process. Let us use word $wbuau'$ to represent $\mathcal{M}$ where $w$, $u$ and $u'$ are three words on alphabet $\mathcal{A}$, $a = l(f)$ and $b = l(e)$. Since $[e, f]$ is a race of $\mathcal{M}$, all events after $e$ (including $e$) on $P_i$ are not enforced before $f$. Therefore, $ua$ represents a scenario $M'$ which is well-formed. According to Definition 5.4, $L(M')|\mathcal{A}_i \notin pref(L(\mathcal{M})|\mathcal{A}_i)$. $M'$ is thus a Type 1 problematic test scenario. This gives a contradiction as required.

For the converse direction, let us suppose that $\mathcal{M}$ does not contain a race $[e, f]$ of $\mathcal{M}$ on a system process $P_i$ such that $msg^{-1}(f)$ is on a user process; we then show that $\mathcal{T}$ has no Type 1 problematic test scenarios. We will use proof by contradiction: assume that the test case of $\mathcal{M}$ causes a Type 1 problematic test scenario which can be represented by $wa$ where $a = l(f)$. According to the definition of Type 1 problematic test scenarios, $f \in R$ happens on a system process $P_i$ such that $w$ is a prefix of $L(\mathcal{M})$ but well-formed $wa$ is not. Notice that every MSC $M$ in $\mathcal{M}$ that contains $f$ and has prefix $w$ can be represented by a well-formed and complete word $wuau'$, where $u$ and $u'$ are two words on alphabet $\mathcal{A}$. We then have $wuau'|\mathcal{A}_i \in L(\mathcal{M})|\mathcal{A}_i$. In addition, since $wa \notin pref(L(\mathcal{M}))$, $wua|\mathcal{A}_i \neq wa|\mathcal{A}_i$ and so $u|\mathcal{A}_i \neq \varepsilon$. This means that in $M$ there is at least an event $x$ on $P_i$ that happens immediately after the event sequence corresponding to $w|\mathcal{A}_i$ but before $f$. Because $wa$ is well-formed, $msg^{-1}(f)$ corresponds to a letter in $w$ and so we have that $x \not<^* msg^{-1}(f)$ and $x \neq msg^{-1}(f)$. Therefore $[x, f]$ is a race of $M$. Because $wa|\mathcal{A}_i \notin pref(L(\mathcal{M})|\mathcal{A}_i)$, $[x, f]$ is a race of $\mathcal{M}$. This provides a contradiction as required. □

This proposition says that only a special type of race leads to controllability problems. The race $[e, f]$ should

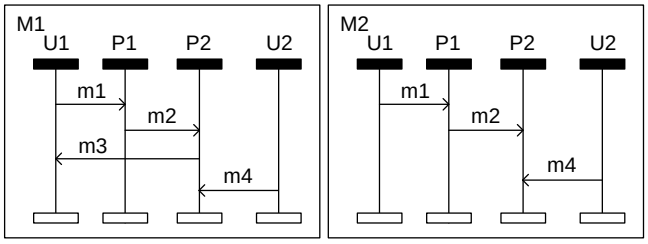


**FIGURE 9.** Controllability problems of timing and observabilities

happen on a system process and event $f$ is the receiving of message $m$ which is sent from a user process.

Another interesting observation is that sometimes testers have the ability to avoid Type 1 problematic test scenarios.

**Example** (**Avoidable controllability problem of timing**) Consider the MSC $M1$ given in Figure 9 in which there is a race $[!m3, ?m4]$ on $P2$. This race causes a controllability problem for $U2$ with local observability since $U2$ cannot guarantee that $m4$ arrives after the sending of $m3$. However, with tester observability, this problem can be resolved for $U2$ by waiting for the observation of $?m3$. This is because $?m3$ happens on $U1$ and events on $U1$ can be observed by $U2$ according to tester observability. Waiting for $?m3$ introduces an enforced order between $!m3$ and $?m4$, so no problematic test scenarios will happen. □

This example shows that a controllability problem caused by a race $[e, f]$ can be overcome by testers if $[e, f]$ satisfies the following conditions: there is an event $x$ which is after $e$ in the enforced order; $x$ is not after the sending event $msg^{-1}(f)$ in the visual order; and $x$ can be observed by the tester. In other words, the technique actually prunes the Type 1 problematic test scenarios by introducing enforced orders between tester observable events.

However, not all controllability problems can be avoided with tester observability.

**Example** (**Unavoidable controllability problem of timing**) Considering MSC $M2$ given in Figure 9 in which $[?m2, ?m4]$ is a race on $P2$. This race causes a controllability problem that is not avoidable with tester observability. $U2$ still cannot decide when to send $m4$ since $?m2$ is a system event which is not observable for $U2$ with tester observability. □

We define avoidable controllability problems of timing as follows.

DEFINITION 5.5. (**Avoidable Controllability Problems of Timing**) *The controllability problems of timing caused by race $[e, f]$ are avoidable if there are two observable events $e'$ and $f'$ where $e' \not\ll^* f'$ such that we have $e \ll^* f$ after we introduce enforced order between $e'$ and $f'$.*

Based on Definition 5.5, we have the following proposition.

PROPOSITION 5.2. *The controllability problems of timing caused by race $[e, f]$ can be avoided if any event in $\{e\} \cup \{x \in E : e \ll^* x \wedge msg^{-1}(f) \not\ll^* x\}$ is observable.*

*Proof.* Let the observable event in $\{e\} \cup \{x \in E : e \ll^* x \wedge msg^{-1}(f) \not\ll^* x\}$ be $y$. An enforced order between $y$ and $msg^{-1}(f)$ can be introduced since both events are tester observable. This leads to $e \ll^* f$ since $y \ll^* msg^{-1}(f)$. Therefore, $e$ is no longer a race with $f$. The proposition is established. □

Based on Proposition 5.2, if testers have global observability, we have the following proposition.

PROPOSITION 5.3. *All controllability problems of timing can be avoided with global observability.*

*Proof.* This proposition follows from Proposition 5.2 and the fact that all events are observable with global observability. □

### 5.3. Algorithms

We provide two algorithms to solve the following problems: detecting races that cause controllability problems of timing in an MSC test case (Algorithm 1); if there are controllability problems of timing, deciding whether they can be avoided (Algorithm 2).

Our algorithms are based on the race detection algorithm for an MSC [4]. The original algorithm calculates the enforced order between any two events in an MSC $M$ with $n$ events. The result is stored in an $n \times n$ matrix $C$. Its first step is generating an index for each of the $n$ events such that the numbering of the events defines a total order which is consistent with the visual order. In our algorithms, we will reuse the index generated for each event in the original algorithm.

Algorithm 1 first constructs automata for process languages for checking whether races of member MSCs are races of the specification. In the beginning of the outer level loop, the algorithm detects all races in each member MSC and stores them in $R$ (Line 4), then the races of an MSC that are on system process and have the second event sent from a user process are picked for further validation (Line 6). Function *isRaceOfSpec* in Line 7 is used to check whether the race of a member MSC is a race of the specification. This is necessary because Definition 5.4 rules out the races of member MSCs which do not cause problematic scenarios. Function *isRaceOfSpec* can be achieved as follows. Let us suppose that $[e, f]$ is a race on $P_i$ based on $M$ of specification $\mathcal{M}$. A word $ub$ can be constructed, where $b = l(f)$ from $M$. If $ub$ is not a prefix of $pref(L(\mathcal{M})|\mathcal{A}_i)$, *isRaceOfSpec* returns *true* otherwise *false*.

The correctness of Algorithm 1 is stated in the following proposition.

```
Input: M
Output: R'
/*Races that lead to controllability
  problems.                              */
1  Initialise R'
2  Construct N /*A set of automata, each
     N_i ∈ N corresponds to process
     languages L(M)|A_i                   */
3  forall M ∈ M do
       /*R is the set of races based on
         M detected using the original
         algorithm in [4]. In the
         process, matrix C, giving the
         enforced order, and indexes of
         events are stored for further
         usage.                           */
4      R = detectOriginalRaces()
5      forall [e, f] ∈ R do
6          if (msg^-1(f) on a user process) &&
           (e on system process) then
7              if isRaceOfSpec((a, b))/*Check
                 if [e, f] is a race of M */
8              then
                   /*Insert the race causing
                     controllability
                     problems into R'     */
9                  insert ([e, f], R')
10             end
11         end
12     end
13 end
```

**Algorithm 1**: Detecting races that cause controllability problems

PROPOSITION 5.4. *Given an MSC specification $\mathcal{M}$ with a finite set of events, the output $R'$ of Algorithm 1 is the set of races which cause controllability problems of timing. The output $R'$ is empty if $\mathcal{M}$ is free of controllability problems of timing.*

*Proof.* We use a two-step approach to prove the correctness of Algorithm 1. We first prove the termination of Algorithm 1. Second, we prove the output is the set of races that cause controllability problems of timing.

For the first step, functions *detectOriginalRaces* and *isRaceOfSpec* called by Algorithm 1 terminate since $\mathcal{M}$ contains a finite number of events. The two loops of Algorithm 1 are controlled by the number of member MSCs in $\mathcal{M}$ and the number of race in $\mathcal{M}$, so the two loops will only executed a finite number of time. Therefore, the first step is established.

For the second step, the output $R'$ contains all the races causing controllability problems of timing. This is based on the fact that Algorithm 1 is designed following Proposition 5.1. The correctness of output follows Proposition 5.1. □

The following proposition states the computational complexity of Algorithm 1.

PROPOSITION 5.5. *Given an MSC specification $\mathcal{M}$ with $l$ MSCs and each with at most $n$ events, the computational complexity of Algorithm 1 is $O(ln^3)$.*

*Proof.* The computational complexity of *detectOriginalRaces* is $O(n^2)$ [4]. According to the definition of race in an MSC, the upper bound on the number of races in $M$ is $n^2$. Process languages of MSC specifications with a finite number of member MSCs can be represented as automata without cycles. It is clear that the construction of $\mathcal{N}$ is linear in the total number of events. Function *isRaceOfSpec* can thus be implemented by checking whether $ub$, where $b = l(f)$, is a prefix of a word which is accepted by the corresponding automaton in $\mathcal{N}$. The complexity of this step is linear in the length of $ub$ which is $O(n)$ since the upper bound of the length is $n$. Therefore, the computational complexity of the inner loop is $O(n^2 \times n) = O(n^3)$. The computational complexity of the algorithm inside the outer loop is $O(n^3) + O(n^2) = O(n^3)$. Therefore, the computational complexity of the outer loop is $O(ln^3)$ since there are $l$ MSCs in $\mathcal{M}$. The overall computational complexity is therefore $O(ln^3)$. The proposition is established. □

It is clear that controllability problems of timing cannot be avoided under local observability and all controllability problems caused by races in $R'$ can be avoided under global observability. Algorithm 2 checks whether controllability problems caused by races in $R'$ can be avoided under tester observability.

Algorithm 2 first calculates the transitive closures of visual order of every member MSC with the same technique used in the original algorithm for race detection in MSCs [4] and stores this in $V$ (Line 2). Lines 5-20 check whether the controllability problems caused by race $[e, f]$ can be avoided according to Proposition 5.2. It checks whether there is an observable event enforced after $e$ and visually not behind $msg^{-1}(f)$, including $e$. If there is such an event, the controllability problems caused by $[e, f]$ can be avoided otherwise it will be added into $R''$. Therefore, if $R''$ is empty at the end, it means that all controllability problems are avoidable with the given observability. The computational complexity of Algorithm 2 is stated in the following proposition.

PROPOSITION 5.6. *Given an MSC specification $\mathcal{M}$ with $l$ MSCs, each having at most $n$ events, the computational complexity of Algorithm 2 is $O(ln^2+n^3)$.*

*Proof.* The complexity of computing each transitive closure of visual order $V$ is $O(n^2)$ using the same approach to calculate the transitive closure of the enforced order [4]. Therefore, the computational complexity of Line 2 is $O(ln^2)$ since there are $l$ member MSCs. The upper bound on the number of races

**Input**: $R'$
/*$R'$ is the set of races detected by Algorithm 1 */
**Output**: $R''$
/*Races that lead to controllability problems that cannot be avoided. */
1 Initialise $R''$
2 calculate and store $V$ for each $M \in \mathcal{M}$
/*$V$ is the transitive closure of visual order of an MSC */
3 **forall** $[e, f] \in R'$ /* $i$ and $j$ are the indexes of $e$ and $msg^{-1}(f)$, respectively. */
4 **do**
5 | **if** *e is not observable* **then**
6 | | isObservable = false
7 | | **forall** $x \in E$ /* $k$ is the index of $x$ */
8 | | **do**
9 | | | **if** $C[i][k] = true$ && $V[j][k] \neq true$/*$C$ is a matrix which stores the enforced order of $M$ calculated in Algorithm 1 */
10 | | | **then**
11 | | | | **if** *x is observable* **then**
12 | | | | | isObservable = true
13 | | | | | break
14 | | | | **end**
15 | | | **end**
16 | | **end**
17 | | **if** *isObservable = false* **then**
18 | | | insert($[e, f]$, $R''$)
19 | | **end**
20 | **end**
21 **end**

**Algorithm 2**: Determining whether controllability problems of timing can be avoided

in $R'$ is $n^2$ and the upper bound of the number of events that have to be checked in the inner loop (Line 7-16) is $n$. The computational complexity of the loop is thus $O(n^2 \times n) = O(n^3)$. Therefore, the overall computational complexity is $O(ln^2 + n^3)$. The proposition is established. □

## 6. CONTROLLABILITY PROBLEMS OF CHOICE

In this section we first define controllability problems of choice and non-local choice in MSC specifications. We then derive the relationship between controllability problems of choice and non-local choice. Finally, an algorithm is provided to detect non-local choices that cause controllability problems of choice.

The Type 2 problematic test scenarios are caused by testers making particular decisions on choices. Therefore, we call them controllability problems of choice.

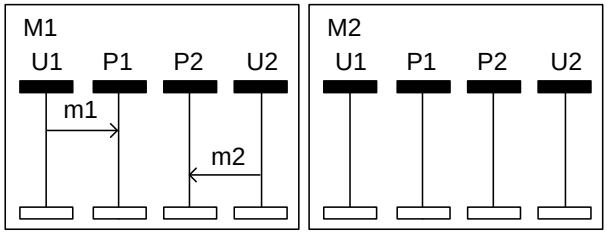


**FIGURE 10.** Non-local choice with terminations

DEFINITION 6.1. (**Controllability Problems of Choice**) *Given an MSC specification $\mathcal{M}$, its MSC test case $\mathcal{T}$ has controllability problems of choice if testers may lead to Type 2 problematic test scenarios.*

### 6.1. Non-local choices

A non-local choice can be described as a choice that depends on information from other processes, but the information is not accessible due to the *local* assumption. Here, the local assumption is that a process is prevented from directly accessing the status of other processes. Generally, a choice is between two event labels. There is a special case that a choice may happen between the termination of a process and an event label.

**Example** (**Non-local choice with a termination of process**) Consider MSC specification $\mathcal{M}5$ with two member MSCs $M1$ and $M2$ shown in Figure 10. Both user processes have problems in deciding whether to send a message or terminate the processes directly. For instance, $U2$ should send $m2$ if $U1$ has sent $m1$. However, $U2$ has no observability of the events on $U1$, so there is a non-local choice between $!m2$ and termination of $U2$. □

To explicitly describe choices between the termination of a process and other events, we include terminations into the alphabet of MSC languages [31] and so $L'$ is used to denote the MSC language extended with an alphabet of terminations of processes, represented by $\downarrow_{\mathcal{P}} = \{\downarrow_i : P_i \in \mathcal{P}\}$. The formal definition of non-local choices based on Definition 2.4 is as follows.

DEFINITION 6.2. (**Non-local Choice** [31]) *Given an MSC specification $\mathcal{M}$, a* non-local choice *is a choice $(w, x, y)$ on $P_i$, such that there exists a word $v \in pref(L'(\mathcal{M}))$, where $v|\mathcal{A}_i = w$, $vx \in pref(L'(\mathcal{M}))$, $vy$ is well-formed and $vy \notin pref(L'(\mathcal{M}))$.*

It has been shown that a non-local choice results in implied scenarios [31].

DEFINITION 6.3. (**Implied Scenarios** [31]) *$w$ represents an* implied scenario *of $\mathcal{M}$ if $w$ is a well-formed word and for each $w|\mathcal{A}_i$ $i \in [n]$, a word $v \in$*

*pref*$(L'(\mathcal{M}))$ *exists such that* $w|\mathcal{A}_i = v|\mathcal{A}_i$, *but* $w \notin$ *pref*$(L'(\mathcal{M}))$.

The concept of implied scenarios was identified in the research on synthesising automata from MSCs [48].[5] In [48], an MSC specification $\mathcal{M}$ with a finite number of member MSCs is said to be *safely realisable* if and only if there exists a synthesised model whose behaviour contains no implied scenarios. It is clear that we have also the following proposition.

PROPOSITION 6.1. *An MSC specification has no implied scenarios if and only if it is safely realisable.*

The relationship between non-local choice and implied scenarios can be established.

PROPOSITION 6.2. *An MSC specification is non-local choice free if and only if it does not lead to implied scenarios.*

*Proof.* According to Proposition 23 in [31], we know that an MSC specification is non-local choice free if and only if it is safely realisable. Moreover, Proposition 6.1 shows that an MSC specification has no implied scenarios if and only if it is safely realisable. The proposition is established. □

By comparing the definition of Type 2 problematic test scenarios and implied scenarios, it is clear that Type 2 problematic test scenarios are implied scenarios with the restriction that the first event violating the specification is on a user process and is a send event.

### 6.2. Non-local choice and controllability problems

It appears that controllability problems of choice are actually caused by non-local choices in MSC specifications.

**Example** (**Non-local choice and controllability problems**) Consider again an MSC specification $\mathcal{M}1$ with two member MSCs, $M1$ and $M2$ given in Figure 2. There are non-local choices on $U1$ and $U2$ according to Definition 6.2. $(\varepsilon, ?m4, !m1)$ is a non-local choice on $U1$ since we can find $!m3?m3!m4?m4 \in$ *pref*$(L(\mathcal{M}1))$ but $!m3?m3!m4!m1 \notin$ *pref*$(L(\mathcal{M}1))$; $(\varepsilon, ?m2, !m3)$ is a non-local choice on $U2$ since we can find $!m1?m1!m2?m2 \in$ *pref*$(L(\mathcal{M}1))$ but $!m1?m1!m2!m3 \notin$ *pref*$(L(\mathcal{M}1))$. These non-local choices lead to an implied scenario, $M3$ given in Figure 2, according to Proposition 6.2. Under local observability, $M3$ is a problematic test scenario. □

However, some non-local choices do not lead to controllability problems. Obviously, non-local choices on system processes do not cause controllability problems.

[5]The definition of implied scenarios in this paper involves deadlock scenarios described in [48].

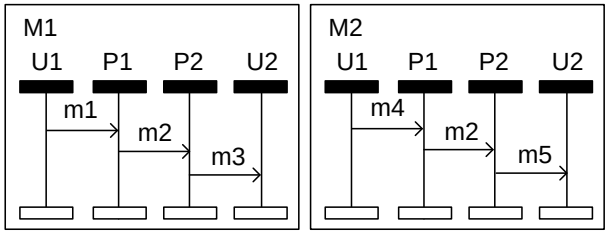


**FIGURE 11.** Non-local choice not leading to controllability problems

**Example** (**Non-local choice on user process**) MSC specification $\mathcal{M}6$ given in Figure 11 contains two MSCs $M1$ and $M2$. There is a non-local choice on $P2$: $(?m2, !m3, !m5)$ since $!m1?m1!m2?m2!m3 \in$ *pref*$(L(\mathcal{M}6))$ but $!m4?m4!m2?m2!m3 \notin$ *pref*$(L(\mathcal{M}6))$, but it is on a system process. □

Non-local choice $(w, x, y)$ causes no controllability problems of choice if $y \in R$ in a non-local choice $(w, x, y)$. This is because the tester does not control the receiving of a message.

**Example** Reconsider MSC specification $\mathcal{M}3$ given in Figure 6 with two MSC $M1$ and $M2$. There is a non-local choice $(\varepsilon, ?m1, ?m2)$ on $U1$ and it leads to a implied scenario $M3$ shown in the same figure. However, this non-local choice does not cause controllability problems of choice because it involves $U1$ receiving a message. □

Whether a non-local choice causes controllability problems also depends on the type of observability applied.

**Example** (**Controllability problems of choice only with local observability**) Reconsider MSC specifications $\mathcal{M}1$ and $\mathcal{M}2$ in Figures 2 and 5. For controllability problems in $\mathcal{M}1$, once $U1$ or $U2$ sends the first message, the other tester is aware of this with tester observability. For $\mathcal{M}2$, $U2$ knows what has been sent from $U1$ to $P1$, so it will choose the right message to send accordingly. □

In fact, the problematic test scenarios under local observability may be automatically ruled out by the test cases generated with tester observability.

**Example** (**Problematic test scenarios avoided by tester observability**) Consider MSC specification $\mathcal{M}1$ in Figures 2. Let the process index for $U1$ and $U2$ be 1 and 2, the test case is thus $(L_1, L_2)$ and $L_1 = L_2 = \{!m1?m2, !m3?m4\}$ with tester observability. At the beginning of a test run, $U1$ and $U2$ both can choose to send messages. However, once one of the testers sends a message, the other tester is restricted to receiving the corresponding message. This is because, for example, after $U1$ sends $m1$, $U2$ cannot send $m3$ otherwise it will violate the test case generated with tester observability. □

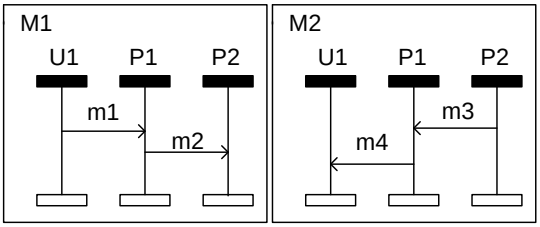


**FIGURE 12.** A controllability problems of choice with tester observability

There are non-local choices that cause controllability problems of choice with tester controllability.

**Example** (**Controllability problems of choice with tester observability**) Consider an MSC specification $\mathcal{M}7$ given in Figure 12. This specification is similar to $\mathcal{M}1$ given in Figure 2. The difference is that the right most process in $\mathcal{M}7$ is a system process. With tester observability, the tester cannot observe events on $P2$, so after $P2$ sends $m3$, the choice to send $m1$ is still open for tester $U1$. Therefore, the non-local choice on $U1$, $(\varepsilon, ?m4, !m1)$, may lead to an implied scenario $!m3!m1$ which is also a problematic test scenario. □

To conclude, we have the following proposition.

PROPOSITION 6.3. *The test case $\mathcal{T}$ of an MSC specification $\mathcal{M}$ has controllability problems of choice if and only if there is a non-local choice $(u, x, y)$ on user process $P_i$ and $y \in \mathcal{A}^{\mathcal{S}}$ such that the non-local choice leads to an implied scenario $wy$ such that $w|\mathcal{A}_i = u$ and $wy|A^i \in pref(L_i)$.*

*Proof.* For the forward direction, let us suppose that the test case has controllability problems of choice. We then show that there is a non-local choice $(u, x, y)$ on a user process $P_i$ and $y \in \mathcal{A}^S$ and $(u, x, y)$ leads to an implied scenario $wy$ such that $wy|A^i \in pref(L_i)$. The test case having controllability problems of choice means that it causes a Type 2 problematic test scenario according to Definition 6.1. Let us use $wy \notin pref(L(\mathcal{M}))$ to represent the problematic test scenario. We have $wy|A^i \in pref(L_i)$ according to Definition 4.1. This implies that $wy|\mathcal{A}_i \in pref(L(\mathcal{M}))$. Therefore, $wy$ is an implied scenario according to Definition 6.3. In addition, according to Proposition 6.2, there is a non-local choice on $P_i$ that can be represented as $(u, x, y)$, where $x \in \mathcal{A}_i$ for implied scenario $wy$. Finally, by Definitions 6.1 and 4.1, $y$ is sending of a message and in alphabet $\mathcal{A}^S$. The forward direction is established.

For the converse direction, it is clear that an implied scenarios $wy$ can be constructed from the non local choice where $w|\mathcal{A}_i = u$. $wy$ is a problematic test scenario since $wy \notin pref(L(\mathcal{M}))$ and $y \in \mathcal{A}^S$. Therefore, a non-local choice on a user process causes a controllability problem. □

The proposition states that only non-local choices on user processes such that the tester observations are consistent with the specification will cause controllability problems. In addition, the condition $y \in \mathcal{A}^{\mathcal{S}}$ means that the non-local choice should involve a sending of a message. This proposition applies with all notions of observability since $A^i$ changes with corresponding observability. In particular, we have the following proposition with global observability.

PROPOSITION 6.4. *No non-local choice causes controllability problems with global observability.*

*Proof.* This follows from Proposition 6.3 and the fact that all events are observable with global observability. □

### 6.3. Algorithm

We can now derive an algorithm to check whether a test case is free of controllability problems of choice and detect non-local choices that cause controllability problems of choice. This algorithm is developed based on the non-local choice detection algorithm in [31]. The algorithm in [31] was an extension of the algorithm for checking whether an MSC specification is safely realisable [48]. After running the algorithm in [31], a set of non-local choices $N$ is found for the given MSC specification $\mathcal{M}$. Algorithm 3 checks the set of non-local choice based on Proposition 6.3. The input of Algorithm 3 is $\mathcal{M}$. The output is the set of non-local choices which lead to controllability problems of choice in the test case generated with a given type of observability. The test case $\mathcal{T}$ of MSC specification $\mathcal{M}$ is free of controllability problems of choice if the output is empty.

Given an MSC specification $\mathcal{M} = \{M_i : 0 < i \leq k\}$ with $k$ MSCs, a non-local choice $(w, x, y)$ on $P_j$ is actually formed by two member MSCs $M_s$ and $M_t$ where $0 < s < t \leq k$. $w$ is the common prefix on $P_j$ of $M_s$ and $M_t$. After $w$, $x$ and $y$ describe two possible behaviours according to the process language $L(\mathcal{M})|\mathcal{A}_i$. At line 4 of the algorithm, two partial MSCs $M'_s$ and $M'_t$ are recovered from $x$ and $y$. $M'_s$ contains all events visually before $x$ in $M_s$ and $M'_t$ contains all events visually before $y$ in $M_t$. Let $E_d$ be the set of events in $M_s$ but not in $M_t$, we thus can decide whether the non-local choice causes controllability problems by checking whether there is an event in $E_d$ that is observable according to the corresponding observability. If an event in $E_d$ is observable, then the non-local choice does not cause a controllability problem.

We have following proposition to state the computational complexity of Algorithm 3.

PROPOSITION 6.5. *Given an MSC specification with $r$ events, $k$ processes, $l$ MSCs, each with at most $n$ events and containing $q$ non-local choices, the computational complexity of Algorithm 3 is $O(l^2k + rk + qn)$.*

*Proof.* We know that it is possible to detect the non-local choices in an MSC specification with $l$ MSCs, $k$

**Input**: $\mathcal{M}$
**Output**: $N'$ /*non-local choices that lead to controllability problems.*/
1 Initialise $N'$
2 $N$ = detectNonLocalChoice() /*$N$ is the set of non-local choices found using the algorithm in [31] */
3 **forall** $(w, x, y) \in N$ *with* $y \in \mathcal{A}^S$ **do**
4 Compute partial MSCs $M'_s$ and $M'_t$
5 $leadToCP = true$;
6 **forall** $e \in E_d$ /* $E_d$ is the set of events that appear in $M'_t$ but not in $M'_s$ */
7 **do**
8 **if** $e$ *is observable* **then**
9 $leadToCP = false$
10 **end**
11 **end**
12 **if** $leadToCP = true$ **then**
/*Insert the non-local choice causing controllability problems into $N'$ */
13 insert $((w, x, y), N')$
14 **end**
15 **end**

**Algorithm 3**: Justify whether non-local choices cause controllability problems

processes and $r$ events in time $O(l^2k + rk)$ [48, 31]. The computational complexity of Algorithm 3 depends on the number of events in $E_d$. We know that the upper bound of the number is $n$. In addition, from an event such as $x$, the corresponding $M'_s$ can be computed in time linear in the number of events in $M_s$. The upper bound of the number is also $n$. The computational complexity of the loop in Algorithm 3 is thus $O(qn)$ when the number of non-local choice in $N$ is $q$. Therefore, the overall computational complexity of detecting non-local choices that cause controllability problems is $O(l^2k + rk + qn)$ □

## 7. OVERCOMING CONTROLLABILITY PROBLEMS

In practice, different techniques proposed in the literature can be used to overcome controllability problems that occur in MSC-based testing, for example, monitoring systems [49], status messages [51] and a coordination service [46]. In this section, we propose to use both status messages and a coordination service to tackle the controllability problems in MSC-based testing.

The coordination service collects the status information from all testers and, if possible, also from system processes. The coordination service thus can provide direction to testers when they come across controllability problems.

Coordination service $CS$ can be implemented as a component on one of the testers. This makes the tester a coordination tester. $CS$ receives status updates from each process after the changes of status of each tester or system process. Status updates can be implemented as *status messages*. Status messages are sent to $CS$ from testers and SUT processes after every communication event happens. In this way, $CS$ can build an overview of the current test scenario. However, some additional routines need to be developed to help overcoming controllability problems.

### 7.1. Avoiding controllability problems of timing

According to Propositions 5.2 and Proposition 5.3, controllability problems of timing may be avoided if enforced orders can be introduced between specific pairs of observable events in the MSC specification. More specific, let us suppose that race $[e, f]$ causes a controllability problem, if we can introduce an enforced order between event $x$ which is enforced after $e$ but not visually after $msg^{-1}(f)$ (the sending of a message from a tester leading to the receive event $f$), the controllability problem can be overcome.

If $CS$ receives status messages only from testers, a given pair of events, $x$ and $msg^{-1}(f)$, are on two different testers, $T_i$ and $T_j$, respectively. The enforced order between the two events can be achieved as follows. The tester $T_j$, which will send event $msg^{-1}(f)$, waits until it receives permission from $CS$; $CS$ sends permission once it has received the update that event $x$ has happened on the other tester $T_i$. The correctness of using $CS$ to overcome avoidable controllability problems of timing is stated in Proposition 7.1. We note that the proofs of correctnesses of overcoming other controllability problems are similar.

PROPOSITION 7.1. *Let us suppose that $\mathcal{M}$ is an MSC specification and test case $\mathcal{T}$ for $\mathcal{M}$ has set $C$ of controllability problems of timing. Controllability problems of $\mathcal{M}$ can be avoided by using a coordination service $CS$ which receives update messages and sends coordination messages from testers.*

*Proof.* Given MSC specification $\mathcal{M}$ and its test case $\mathcal{T}$, to prove that controllability problems of timing can be avoided by coordination service $CS$, we consider $CS$ as an additional user process. A new MSC specification $\mathcal{M}'$ including process $CS$ is thus formed from $\mathcal{M}$. We then prove the test case $\mathcal{T}'$ of $\mathcal{M}'$ formed considering $CS$ as a tester contains no avoidable controllability problem of timing. We prove the proposition in two steps. First we show that no new controllability problems of timing will be introduced in $\mathcal{T}'$. Second, existing races that causes controllability problems of timing in $\mathcal{T}$ have been removed.

For the first step, messages between $CS$ and other user processes will not introduce any event on system

processes of $\mathcal{M}'$. In addition, the introduction of $CS$ adds events into the partial order of member MSCs of $\mathcal{M}'$, so no enforced order in member MSCs of $\mathcal{M}$ is reduced. Therefore, no race on system processes is added. Therefore, no new controllability problems of timing will be introduced in $\mathcal{T}'$.

For the second step, let $c$ be any controllability problem in $C$ which is caused by a race $[e, f]$ on a system process. Let $x$ be the observable event where $e \ll^* x \ \wedge msg^{-1}(f) \not<^* x$ on tester $T_i$ and $msg^{-1}(f)$ is on $T_j$ since $c$ is avoidable. Let $T_i$ send update $m_x$ to $CS$ after $x$ happens and $CS$ sends a coordination message $m_f$ to $T_j$ after it receives $m_x$. We thus have $x \ll !m_x$, $!m_x \ll ?m_x$, $?m_x \ll !m_f$, $!m_f \ll ?m_f$ and $?m_f \ll msg^{-1}(f)$. Consequently, $e \ll f$ and the pair of events no longer forms a race in $\mathcal{M}'$. Therefore, $e$ and $f$ do not form a controllability problem of timing in $\mathcal{T}'$ and the second step is established. □

If $CS$ also receives status messages from system processes, $x$ can be $e$ in the problematic race $[e, f]$. In this case, $CS$ sends permission to the tester which sends event $msg^{-1}(f)$ once it receives the update that $e$ has happened.

In the routine described above, the premise is that testers and $CS$ know the pairs of $x$ and $msg^{-1}(f)$. This can be achieved by running a variant of Algorithm 2 which calculates the problematic races $[e, f]$. The only modification is that $x$ and $msg^{-1}(f)$ are output after line 12 and this modification does not change the complexity of the original algorithm.

### 7.2. Overcoming controllability problems of choice with better observability

The number of controllability problems of choices can be reduced if testers have better observabilities as explained in Section 6. According to Propositions 6.3 and 6.4, it is clear that there are most controllability problems with local observability, some controllability problems with local observability do not happen with tester observability and there are no controllability problems if testers have global controllability. With the help of $CS$, we can achieve a similar effect on reducing controllability problems of choice as all testers have tester observability or global observability.

Let us suppose that $(u, x, y)$ is a non-local choice on $P_i$ that causes a controllability problem of choice with local observability. $(u, x, y)$ is thus a choice on the tester simulating $P_i$ and $y$ is a send event.

With $CS$ receiving status messages from all testers, the tester simulating $P_i$ can overcome the controllability problems as follows. When the tester comes across non-local choice $(u, x, y)$ and it considers sending message, $y$, which may lead to a problematic test scenario, the tester is required to send a request to $CS$ before it sends $y$. Once $CS$ receives such a request, it checks whether $y$ will lead to a problematic test scenario by constructing $wy$ where $w$ is the word representing the current running test scenario. $w$ is maintained by $CS$ based on previous status messages. With the assumption that an MSC specification contains a finite set of MSCs, checking whether $wy$ represents a problematic test scenario can be solved in time that is linear in the number of events in $wy$. Having checked whether the sending of $y$ will lead to a problematic scenario, $CS$ responds to the request from the tester.

If all system processes send status messages to $CS$, testers can overcome all controllability problems under local observability. Once $CS$ receives a request regarding a problematic non-local choice, $CS$ checks whether it leads to an implied scenario according to the specification.

This solution is based on all non-local choices under local observability being known by testers, so that they can send coordination requests to $CS$. For an MSC specification with a finite number of MSCs, problematic non-local choices under local observability can be efficiently calculated using Algorithm 3.

## 8. DISCUSSION

In discussing controllability problems in previous sections, we assumed that communications are asynchronous and non-FIFO. In fact, it is not difficult to extend the corresponding results to other types of communication if corresponding races and non-local choices are detectable.

Let us consider the controllability problems with FIFO communications as an example. Obviously, it is reasonable to assume that the communications described in the MSC specifications do not violate FIFO rule. Some of the Type 1 problematic test scenarios which may happen with the non-FIFO communication do not happen with the FIFO communication. This is because the FIFO rule introduces additional enforced order pairs among MSC events and these additional orders may reduce races in MSC specifications. Formally, with the FIFO communication, enforced order of an MSC can be defined as follows.

DEFINITION 8.1. (**Enforced Order with FIFO**) *Given an MSC $M$ with set $E$ of events, the transitive closure of the relation, $\ll \doteq \{(x, y) \in <: \ y = msg(x) \vee (y \in S \wedge p(x) = p(y))\} \cup \{(x, y) : x, y \in E \wedge msg^{-1}(x) < msg^{-1}(y) \wedge p(x) = p(y) \wedge p(msg^{-1}(x)) = p(msg^{-1}(y))\}$, is the enforced order of $M$ with the FIFO communication.*

**Example** (**FIFO communication**) $M2$ in Figure 4 does not cause Type 1 problematic test scenarios. With the modified enforced order, events $?m1$ and $?m3$ in $M2$ of Figure 4 do not form a race since $m1$ always arrives at $P1$ before $m3$ with the FIFO communication. □

Races in an MSC specification may change when the nature of communication changes to FIFO, but this

does not invalid Proposition 5.1 since the proposition is based on detected races. In addition, Algorithms 1 and 2 are also applicable. This is because the change of communication only affects function *detectOrininalRace* and the enforced order referred to by the algorithms.

For the controllability problems of choice, we notice that implied scenarios may violate the FIFO rule.

**Example** (**Implied scenarios violating FIFO communication**) Consider MSC specification $\mathcal{M}3$ given in Figure 6, the implied scenario $M3$ violates the FIFO rule. □

However, the type of non-local choices as shown in the example do not lead to Type 2 problematic test scenarios according to Definition 4.1 in which it is required that the problematic option should be a positive behaviour. Actually, we can show that Proposition 6.3 and Algorithm 3 apply to the FIFO communication. Let us suppose that non-local choice $(u, x, y)$ of MSC specification $\mathcal{M}$ causes Type 2 problematic test scenarios. $y \in \mathcal{A}^{\mathcal{S}}$ and a word $w$ can be found such that $wx \in pref(L(\mathcal{M}))$ but $wy \notin pref(L(\mathcal{M}))$. Obviously, $u$ does not violate the FIFO rule because it is a prefix of an MSC that satisfies the FIFO assumption. It is clear that $uy$ does not form a scenario that violates the FIFO rule since $y$ is a send label. Therefore, non-local choice $(u, x, y)$ with $y \in \mathcal{A}^{\mathcal{S}}$ causes Type 2 problematic test scenarios with the FIFO communication. Proposition 6.3 applies to the FIFO communication. Consequently, Algorithm 3 is applicable for checking whether non-local choices in MSC specifications cause controllability problems of choice with the FIFO communication.

## 9. CONCLUSIONS

This paper investigated controllability problems that might happen when testing distributed system based on MSC specifications. The analysis was based on an MSC conformance testing architecture in which there are multiple testers and system processes. In addition, three different types of observability (local, tester and global) were considered. The results could be used when analysing an MSC specification of the SUT, where we wish to test based on the entire specification, or when considering test cases defined using MSCs. Both situations are highly relevant since MSCs (and SDs) are widely used to model distributed systems but also to describe test cases.

We showed that there are only two types of controllability problems: controllability problems of timing and controllability problems of choice. We showed that one important type of pathology of MSC, race, causes controllability problems of timing and another type of pathology, non-local choice, causes controllability problems of choice. Moreover, not all races and non-local choices lead to controllability problems. In addition, some controllability problems of timing with local observability can be avoided with tester observability and all controllability problems of timing can be avoided if testers have global observability. For controllability problems of choice, we showed that some non-local choices cause problems with local observability but not with tester observability; if testers have global observability, there are no controllability problems of choice in MSC test cases.

With the given relationships between MSC pathologies and controllability problems, algorithms were derived to capture controllability problems of timing and choice. We introduced Algorithm 1 to find races that cause controllability problems of timing based on the race detection algorithm in [4]. The result of Algorithm 1 is then input to Algorithm 2 to determine whether the detected controllability problems of timing can be avoided with better observability. The algorithm for controllability problems of choice was derived from two algorithms: one checks the safe realisability of an MSC specification with a finite number of MSCs [48] and the other detects all non-local choices in such an MSC specification [31]. We showed that all of the algorithms had polynomial time complexity. This means that the controllability problems of test cases of MSC specifications with a finite member MSCs can be efficient resolved.

In practice, controllability problems can be overcome by different techniques. We showed that a coordination service with status messages could be used to alleviate the controllability problems in testing with MSCs. In the proposed approach, races and non-local choices that lead to controllability problems were calculated before the testing process and this can be achieved by applying the algorithms provided in this paper. With the knowledge of the existing controllability problems, coordination messages between testers and a coordination service were transmitted in run-time to coordinate the behaviour of testers.

We note that the algorithms apply to MSC specifications with a finite number of MSCs. With more complex MSC specifications such as high-level MSCs (HMSCs) which may have an infinite number of MSCs, the algorithms may need further improvements. It has been shown that detecting races and non-local choices in complex MSC specification are EXSPACE-hard problems [28, 52]. We might solve the controllability problems of MSCs in complex MSCs by adding restrictions. For example, restricting the number of unfoldings of the loops in HMSCs specifications. In addition, Interaction Diagrams (IDs) of UML 2.0 have become a popular modelling language in the software industry. It would be worth extending current work to IDs. However, it has been shown that there are significant differences between IDs and MSCs [53, 54]. Finally, an MSC-based testing tool which implements algorithms and coordination services proposed in this paper would have been a nice contribution.

## ACKNOWLEDGEMENTS

This research was partially supported by the UK EPSRC project EP/G04354X/1, The Birth, Life and Death of Semantic Mutants.

## REFERENCES

[1] ITU-T (2004). ITU-T Recommendation Z.120 Message Sequence Chart.

[2] Mauw, S., Reniers, M., and Willemse, T. (2001) Message Sequence Charts in the software engineering process. *Handbook of Software Engineering and Knowledge Engineering*, **1**, 437–464, World Scientific Publishing.

[3] Haugen, Ø. (2001) MSC-2000 interaction diagrams for the new millennium. *Computer Networks*, **35**, 721–732.

[4] Alur, R., Holzmann, G., and Peled, D. (1996) An analyzer for message sequence charts. *Software Concepts and Tools*, **17**, 70–77.

[5] Muscholl, A., Peled, D., and Su, Z. (1998) Deciding properties for message sequence charts. *Proceedings of the 1st Conference on Foundations of Software Science and Computation Structures*, Lisbon, Portugal, pp. 226–242, Springer-Verlag, London.

[6] Hierons, R. M. and et al. (2009) Using formal specifications to support testing. *ACM Computing Surveys*, **41**, 1–76.

[7] Ural, H. and Yang, B. (1991) A test sequence selection method for protocol testing. *IEEE Transactions on Communications*, **39**, 514 –523.

[8] Dalal, S., Jain, A., Karunanithi, N., Leaton, J., Lott, C., Patton, G., and Horowitz, B. (1999) Model-based testing in practice. *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, CA, USA, pp. 285–294, IEEE Computer Society.

[9] Offutt, J., Liu, S., Abdurazik, A., and Ammann, P. (2003) Generating test data from state-based specifications. *Software Testing, Verification and Reliability*, **13**, 25–53.

[10] Tretmans, G. and Brinksma, H. (2003) Torx: Automated model-based testing. *Preceedings of the first european conference on Model-Driven Software Engineering*, Nuremberg, Germany, pp. 31–43.

[11] Petrenko, A., Boroday, S., and Groz, R. (2004) Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering*, **30**, 29–42.

[12] Muccini, H., Inverardi, P., and Bertolino, A. (2004) Using software architecture for code testing. *IEEE Transactions on Software Engineering*, **30**, 160 – 171.

[13] Pretschner, A., Prenninger, W., Wagner, S., Kühnel, C., Baumgartner, M., Sostawa, B., Zölch, R., and Stauner, T. (2005) One evaluation of model-based testing and its automation. *Proceedings of the 27th International Conference on Software Engineering*, New York, NY, USA, pp. 392–401, IEEE Computer Scoiety.

[14] Campbell, C., Grieskamp, W., Nachmanson, L., Schulte, W., Tillmann, N., and Veanes, M. (2005) Model-based testing of object-oriented reactive systems with spec explorer. Technical Report MSR-TR-2005-59.

[15] Shousha, M., Briand, L., and Labiche, Y. (2010) A UML/MARTE model analysis method for uncovering scenarios leading to starvation and deadlocks in concurrent systems. *IEEE Transactions on Software Engineering*, **Preprint**, 1–54.

[16] Grieskamp, W., Kicillof, N., Stobie, K. and Braberman, V. (2011) Model-based quality assurance of protocol documentation: tools and methodology. *The Journal of Software Testing, Verification and Reliability*, **21**, 55–71.

[17] Baker, P., Bristow, P., Jervis, C., King, D., and Mitchell, B. (2003) Automatic generation of conformance tests from message sequence charts. *Telecommunications and beyond: The Broader Applicability of SDL and MSC*, Lecture Notes in Computer Science, **2599**, pp. 170–98. Springer-Verlag.

[18] Chung, I. S., Kim, H. S., Bae, H. S., Kwon, Y. R., and Lee, B. S. (1999) Testing of concurrent programs based on message sequence charts. *Proceedings of the 3rd International Symposium on Software Engineering for Parallel and Distributed Systems*, Los Alamitos, CA, USA 72–82, IEEE Computer Scoiety.

[19] Koch, B., Grabowski, J., Hogrefe, D., and Schmitt, M. (1999) Autolink – a tool for automatic test generation from SDL specifications. *Proceedings of the 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*, Boca Raton, FL, USA, 10, pp. 114–25, IEEE Computer Scoiety.

[20] Grabowski, J., Hogrefe, D., and Nahm, R. (1993) Test case generation with test purpose specification by MSCs. *Proceedings of the 6th SDL Forum*, Darmstadt, Germany, pp. 253–65.

[21] Boroday, S., Petrenko, A., and Ulrich, A. (2009) Implementing MSC tests with quiescence observation. *Proceedings of the 21st International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop*, Berlin, Heidelberg, pp. 49–65, Springer Berlin, Heidelberg.

[22] Lee, D. and Yannakakis, M. (1996) Principles and methods of testing finite state machines - a survey. *Proceedings of the IEEE*, **84**, 1090–1123.

[23] Petrenko, A. (2001) Fault model-driven test derivation from finite state models: Annotated bibliography. *Modeling and Verification of Parallel Processes*, Lecture Notes in Computer Science, **2067**, pp. 196–205. Springer Berlin, Heidelberg.

[24] Brinksma, H. and Tretmans, G. (2001) Testing transition systems: An annotated bibliography. *Modeling and Verification of Parallel Processes*, Lecture Notes in Computer Science, **2067**, pp. 187–195. Springer Berlin, Heidelberg.

[25] Tretmans, G. (2008) Model based testing with labelled transition systems. In Hierons, R. M., Bowen, J. P., and Harman, M. (eds.), *Formal Methods and Testing*, Lecture Notes in Computer Science, **4949**, pp. 1–38. Springer.

[26] Hélouët, L. (2001) Some pathological message sequence charts, and how to detect them. *Proceedings of the 10th International SDL Forum*, Copenhagen, Denmark, 1, pp. 348–364.

[27] Mitchell, B. (2005) Resolving race conditions in asynchronous partial order scenarios. *IEEE Transactions on Software Engineering*, **31**, 767–784.

[28] Muscholl, A. and Peled, D. (1999) Message sequence graphs and decision problems on Mazurkiewicz traces. *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, London, UK, pp. 81–91, Springer Berlin, Heidelberg.

[29] Ben-Abdallah, H. and Leue, S. (1997) Syntactic detection of process divergence and non-local choice in message sequence charts. *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Enschede, Netherlands, pp. 259–274, Springer Berlin, Heidelberg.

[30] Uchitel, S., Kramer, J., and Magee, J. (2004) Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Transactions on Software Engineering and Methodology*, **13**, 37–85.

[31] Dan, H., Hierons, R. M., and Counsell, S. (2010) Non-local choices and implied scenarios. *Proceedings of the 8th International Conference on Software Engineering and Formal Methods*, Pisa, Italy, pp. 53–62, IEEE Computer Scoiety.

[32] Dan, H., Hierons, R. M., and Counsell, S. (2011) A Framework for Pathologies of Message Sequence Charts. *submitted*, **N/A**.

[33] Dan, H. and Hierons, R. M. (2011) Conformance testing from Message Sequence Charts. *Proceedings of the 4th International Conference on Software Testing, Verification and Validation*, Berlin, Germany, pp. 279–288, IEEE Computer Scoiety.

[34] Cacciari, L. and Rafiq, O. (1999) Controllability and observability in distributed testing. *Information and Software Technology*, **41**, 767–780.

[35] Chen, W. and Ural, H. (1995) Synchronizable checking sequences based on multiple UIO sequences. *IEEE/ACM Transactions on Networking*, **3**, 152–157.

[36] Dssouli, R. and von Bochmann, G. (1985) Error detection with multiple observers. *Proceedings of the IFIP WG6.1 Fifth International Conference on Protocol Specification, Testing and Verification V*, Toulouse-Moissac, France, pp. 483–494, North-Holland, Amsterdam, The Netherlands.

[37] Dssouli, R. and von Bochmann, G. (1986) Conformance testing with multiple observers. *Proceedings of the IFIP WG6.1 Sixth International Conference on Protocol Specification, Testing and Verification VI*, Montreal, Canada, pp. 217–229, North-Holland, Amsterdam, The Netherlands.

[38] Haar, S., Jard, C., and Jourdan, G.-V. (2007) Testing input/output partial order automata. *Proceedings of the 19th International Conference on Testing of Communicating Systems*, Tallinn, Estonia, pp. 171–185, Springer Berlin, Heidelberg.

[39] Hierons, R. M. and Ural, H. (2008) The effect of the distributed test architecture on the power of testing. *The Computer Journal*, **51**, 497–510.

[40] Hierons, R. M., Merayo, M. G., and Núñez, M. (2008) Implementation relations for the distributed test architecture. *Proceedings of the 20th International Conference on Testing of Software and Communicating Systems*, Tokyo, Japan, pp. 200–215, Springer-Verlag Berlin, Heidelberg.

[41] Sarikaya, B. and v. Bochmann, G. (1984) Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, **32**, 389–395.

[42] Boyd, S. and Ural, H. (1991) The synchronization problem in protocol testing and its complexity. *Information Processing Letters*, **40**, 131–136.

[43] Ural, H. and Williams, C. (2006) Constructing checking sequences for distributed testing. *Formal Aspects of Computing*, **18**, 84–101.

[44] Khoumsi, A. (2002) A temporal approach for testing distributed systems. *IEEE Transactions on Software Engineering*, **28**, 1085–1103.

[45] Hierons, R. M. Controllable testing from nondeterministic finite state machines with multiple ports. *IEEE Transactions on Computers*, **to appear**.

[46] Jard, C., Jéron, T., Kahlouche, H., and Viho, C. (1998) Towards automatic distribution of testers for distributed conformance testing. *Proceedings of IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques and Protocol Specification, Testing and Verification*, pp. 353–368. Kluwer.

[47] Ladkin, P. B. and Leue, S. (1994) Four issues concerning the semantics of Message Flow Graphs. *Proceedings of the 7th International Conference on Formal Description Techniques*, Berne, Switzerland, pp. 355–369, Chapman & Hall, Ltd. London, UK.

[48] Alur, R., Etessami, K., and Yannakakis, M. (2003) Inference of message sequence charts. *IEEE Transactions on Software Engineering*, **29**, 623–633.

[49] Zulkernine, M. and Seviora, R. E. (2002) A compositional approach to monitoring distributed systems. *Proceedings of International Conference on Dependable Systems and Networks*, Bethesda, Maryland, USA, pp. 763–772, IEEE Computer Scoiety.

[50] Baker, P., Bristow, P., Jervis, C., King, D., Thomson, R., Mitchell, B., and Burton, S. (2005) Detecting and resolving semantic pathologies in UML sequence diagrams. *Proceedings of the 10th European Software Engineering Conference held jointly with the 13th International Symposium on Foundations of Software Engineering*, Lisbon, Portugal, pp. 50–59, ACM New York, NY, USA.

[51] Hierons, R. M. (2009) Using status messages in the distributed test architecture. *Information & Software Technology*, **51**, 1123–1130.

[52] Alur, R., Etessami, K., and Yannakakis, M. (2005) Realizability and verification of MSC graphs. *Theoretical Computer Science*, **331**, 97–114.

[53] Dan, H., Hierons, R. M., and Counsell, S. (2007) Thread-based analysis of Sequence Diagrams. *Proceedings of the 27th International Conference on Formal Methods for Networked and Distributed Systems*, Tallinn, Estonia, pp. 19–34, Springer Berlin, Heidelberg.

[54] Dan, H., Hierons, R. M., and Counsell, S. (2007) A Thread-tag based semantics for Sequence Diagram. *Proceedings of the 5th International Conference on Software Engineering and Formal Methods*, London, UK, pp. 173–182, IEEE Computer Scoiety.