The Oracle Problem When Testing from MSCs

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

Message Sequence Charts (MSCs) form a popular language in which scenario-based specifications and models can be written. There has been significant interest in automating aspects of testing from MSCs. This paper concerns the Oracle Problem, in which we have an observation made in testing and wish to know whether this is consistent with the specification. We assume that there is an MSC specification and consider the case where we have entirely independent local testers (local observability) and where the observations of the local testers are logged and brought together (tester observability). It transpires that under local observability the Oracle Problem can be solved in low-order polynomial time if we use sequencing, loops and choices but becomes NP-complete if we also allow parallel components; if we place a bound on the number of parallel components then it again can be solved in polynomial time. For tester observability, the problem is NP-complete when we have either loops or choices. However, it can be solved in low-order polynomial time if we have only one loop, no choices, and no parallel components. If we allow parallel components then the Oracle Problem is NP-complete for tester observability even if we restrict to the case where there are at most two processes.

Keywords: Testing; Oracle Problems; Message Sequence Charts

1. INTRODUCTION

Software testing is an important part of the software development process but is typically expensive, manual, and error prone. This observation has led to significant interest in test automation, with model-based testing (MBT) being one approach to this. In MBT, a model M is produced and testing is then based on this model. For example, test cases might be generated from M and M can be used to help direct testing. An important benefit of MBT is that M can be used as the basis of a *test oracle*: we can check that observations made in testing are consistent with M. Recent work has reported the results of a major industrial project involving hundreds of software engineers, with it being found that the use of MBT led to significant benefits [1]. Most approaches to MBT use behavioural models written in languages that are either state-based or scenario-based. In this paper, we focus on the Oracle Problem when testing from Message Sequence Charts (MSCs): this is the problem of deciding whether a given observation is consistent with an MSC specification.

MSCs form a popular scenario-based language that is suitable for describing the behaviour of distributed systems [2]. MSCs are widely used for requirements analysis and system design in the telecommunications and software industries [3, 4, 5, 6, 7]. MSCs include two forms of graphical models: basic MSCs (bMSCs) and High-level MSCs (HMSCs). Generally, a bMSC is used to represent message interactions among users and multiple processes (sub-systems). An example of a bMSC is given in the left-hand side of Figure 1 in which process P_1 starts a connection request to process P_3. HMSCs are designed for the description of scenarios with complex structures such as choices, loops and even parallel compositions. They therefore allow one to define a, potentially infinite, set of bMSCs. An HMSC can be seen as being a graph that has a start node, termination nodes and MSC nodes which refer to other bMSCs or HMSCs. HMSC H_1 given in the middle of Figure 1 is an example of an HMSC with a choice and

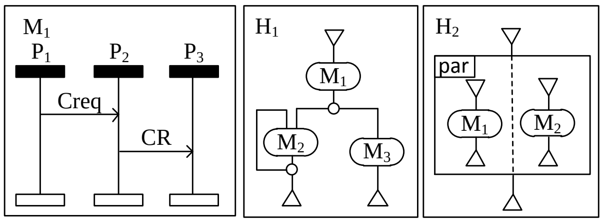

FIGURE 1. Examples of bMSC and HMSC

a loop structure. In the right-hand side of Figure 1, H_2 is an HMSC with a parallel composition in which there are two parallel components that refer to HMSCs.

In the early 1990s, MSCs began to be used in MBT, initially for describing the *test purpose* [8, 9]. A test purpose describes the type of test case required; test cases were then generated in TTCN format[1] from a state-based specification written in System Design Languages (SDL). A tool called *Autolink* [12] was developed not only to implement such an approach [8] but also to generate TTCN test cases directly from MSCs where an SDL model is not available.

In MBT it is necessary to define an implementation relation that states what it means for a behaviour to be allowed by a model and a number of implementation relations have been proposed in existing research on testing from MSCs. Two relatively weak implementation relations for an MSC specification $\mathcal{M}$ containing a set of MSCs were proposed in [13]: *behavioural conformance* and *non-deterministic conformance*. Given an MSC specification $\mathcal{M}$, behavioural conformance is satisfied if and only if for every bMSC $M \in \mathcal{M}$ the SUT can perform a sequence w of events such that w is a linearisation of M. Non-deterministic conformance actually extends behavioural conformance by saying that, if selected pair of events e and f are unordered in $\mathcal{M}$, they must also be unordered in the SUT. An alternative and more intuitive implementation relation requires all observations of the SUT to be consistent with partial orders described in the given MSC specification [14]. Since MSCs are used to describe the interaction between multiple processes, they are often used for distributed systems and a distributed test architecture might then be applied. There are then multiple testers and a range of options regarding the observational power of these testers. This leads to three implementation relations: under *border event conformance* each tester only observes the events that it is directly involved in; under *tester conformance* each tester shares its observations with the other testers during testing; and under *all conformance* each tester observes all of the events, including the exchange of messages by internal processes [15]. In addition to these, Boroday et al. [16] proposed the inclusion of quiescence as an observation.

Another line of research concerned controllability problems, where a tester might not know when to send a message to the SUT because it does not observe the events that involve the other testers [17]. It has been shown that the notion of a controllability problem when testing from an MSC specification is related to MSC pathologies [18]. In this paper, we discuss another critical issue in conformance testing with MSCs: the Oracle Problem. Both the controllability problem and the Oracle Problem need to be addressed in testing a distributed system. Possible controllability problems have to be resolved in running distributed tests so that correct input is sent to the SUT. A test coordinator may help to overcome the controllability problem by sending and receiving additional control messages [18]. We have to solve the Oracle Problem when we have made observations in testing and wish to assign a verdict that states whether a failure has occurred. The Oracle Problem has to be answered with or without a test coordinator. Therefore, the two problems are relevant but different.

There have been several other lines of work regarding MSC-based testing. Lee et al. [19] proposed an approach that first transformed an MSC specification to a state-based model and then tested from the state-based model. In [20], a model checker was used to generated test cases to cover all possible execution paths of an MSC specification. There has also been interest in testing from Live Sequence Charts (LSCs), where the test cases can be generated by interacting with a Play-Engine tool [21, 22]. The *play-out* specifications may lead to a very complex model, so Kugler et al. [23] proposed to use restrictions and execution configurations to simplify the LSC-based testing process. In [24], a TTCN-3 based conformance testing approach was proposed for a broadcast business management system. An extension of MSCs, namely Symbolic Message Sequence Charts (SMSCs), has been proposed; testing from SMSCs required extra steps to generate templates and abstract test cases [25].

Sequence Diagrams (SDs) form a scenario based language that is similar to MSCs and is part of the UML. There are a number of lines of work that investigate testing from SDs [26, 27, 28, 29, 30]. Most of these generate test cases based on both a static model (Class model) and behavioural models or concentrate on specific problems, for example stress testing [27], test input generation for specific parameters [29] and integration testing [30].

This paper makes the following contributions. First, we provide a formal definition of the Oracle Problem for HMSCs (Definition 3.1), adapting it to different types of observability that might be used in testing. We then focus on the computational complexity of the Oracle Problem and we consider a number of restrictions. Towards the end of the paper the results are summarised in Tables 1 and 2. A general pattern is that the computational complexity becomes higher when the testers have greater observational ability (tester observability or global observability) and also if we allow parallel composition. We show that the Oracle Problem is NP-complete for testing from HMSCs even with only choices or loops. We also show that the complexity can be reduced by introducing restrictions such as a bound on the size of the observation set or considering HMSCs which are safely realisable. With either restriction, the Oracle Problem for HMSCs with both choices and loops can be solved in polynomial

[1]TTCN was initially an acronym for Tree and Tabular Combined Notation [10] and later for Testing and Test Control Notation [11].

time. Finally, we show that the complexity is NP-complete for an HMSC with parallel composition and either choice or loop structures even when the testers have local observability. However, this complexity can be reduced to polynomial time by limiting the number of components in parallel compositions.

The Oracle Problem discussed in this paper is essentially caused by parallel/distributed computing and the inability to establish the global order of events. The main motivation for our work is to learn more about the theoretical problems of modelling distributed computing and detecting faults in distributed systems. We believe that this will lead to practical applications in the future. For example, there is a potential to use MSCs to explore synchronisation related faults in distributed software systems. In addition, the results given in the paper can be used to provide practitioners with guidance regarding features that can complicate testing. We believe that such results have the potential to lead to notions of testability for the testing of distributed systems.

The remainder of the paper is organised as follows. In the next section, we give preliminary definitions of MSCs. Section 3 introduces the Oracle Problem for testing from MSCs and formally defines this. In Sections 4 and 5, we investigate the complexity of the Oracle Problem for MSC specifications without or with parallel compositions, respectively. Finally, Section 6 presents conclusions.

2. PRELIMINARIES

In this section we provide definitions of bMSCs and HMSCs and their semantics. The graphical form of bMSCs contains processes and messages 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 [2]. A bMSC thus defines a partial order among the events. The formal representation of bMSCs is as follows.

DEFINITION 2.1. *(**bMSCs**) A* bMSC *M is a tuple $\langle E, C, \mathcal{P}, l, msg, < \rangle$ where: E is a set of events, C is a message alphabet and $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a set of processes; E is partitioned into a set S of send events and a set R of receive events, $E = S \cup R$; $l : E \mapsto \mathcal{A}$ is a labelling function and $\mathcal{A} = \mathcal{A}^S \cup \mathcal{A}^R$ where $\mathcal{A}^S = \{send(i,j,a) : 1 \le i,j \le n \wedge a \in C\}$ is the set of sending of messages and $\mathcal{A}^R = \{receive(i,j,m) : 1 \le i,j \le n \wedge m \in C\}$ is the set of receiving of messages; $send(i,j,m)$ represents the sending of message m from P_i to P_j and $receive(i,j,m)$ the receiving of the corresponding message[2]; $\mathcal{A}_i$ represents the set of labels on process P_i; $msg : S \mapsto R$ is a bijection between send and receive events, matching each send with its corresponding receive; the inverse mapping is $msg^{-1} : R \mapsto S$ between receive and send events; there is 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 $1 \le i \le n$, a total order $<_i$ on the events of process P_i, i.e., on the elements of $p^{-1}(i)$, such that the transitive closure of the relation $< \doteq \bigcup_{1 \le i \le n} <_i \cup \{(s, msg(s)) : s \in S\}$ is a partial order on E, namely, visual order ($<^*$). Note that, since $<_i$ is a total order, it is antisymmetric.*

In terms of testing from MSCs, the trace semantics of bMSCs are particular useful because the observations of testers can be captured by sequences of event labels. The trace semantics can be described as follows [31].

DEFINITION 2.2. *(**Word of bMSC**) Given a bMSC M, a* word *of M is a string $w = w_1 \cdots w_{|E|}$ over $\mathcal{A}$ if and only 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 $1 \le i \le |E|$, $w_i = l(e_i)$.*

Assume that the non-degeneracy condition, also called weak-FIFO, is satisfied.[3] A well-formed and complete word uniquely characterises a bMSC [31]. A word is *well-formed* if all of its receive events have earlier matching sends and a word w is *complete* if all send events have matching receives. A bMSC M describes a set of well-formed and complete words and the set of words is the language of the bMSC, denoted as $L(M)$. We use $pref(w)$ and $pref(L)$ to denote the set of prefixes of word w and the set of prefixes of language L, respectively.

HMSCs is a popular way to organise MSC specifications [32]. HMSCs are formalised as a graph with nodes labelled by bMSCs or other HMSCs as shown in Figure 1. In addition, HMSCs may contain nodes with parallel composition. Events from different components of a parallel composition can interleave with any event from other components, but the only restriction is that the event order within each component will be preserved. The semantics of an HMSC is given by the weak sequencing of the nodes on the paths of the HMSC. Weak sequencing means that the individual processes of bMSCs are concatenated and there is no synchronisation at concatenation points between bMSCs. Therefore, an HMSC defines a set of member bMSCs and the set of bMSCs can be infinite due to the loop structure in the graph. The trace semantics of a complex MSC specification can be defined as follows.

[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]MSC M is non-degenerate if M does not contain two send events $e1$ and $e2$ such that $l(e1) = l(e2)$, $e1 < e2$ and $msg(e2) < msg(e1)$.

Definition 2.3. ***(Semantics of MSC specification)*** *For an MSC specification $\mathcal{M}$[4], the language of $\mathcal{M}$, $L(\mathcal{M})$, is the union of the sets of words of all member bMSCs in $\mathcal{M}$, $L(\mathcal{M}) = \bigcup_{M \in \mathcal{M}} L(M)$.*

MSCs implicitly define process behaviours. The process behaviour can be extracted from MSC specifications by projection.

Definition 2.4. ***(Process language)*** *Given a word w which characterises bMSC M and process P_i in M, $w|P_i$ denotes the projection of w on process P_i. For an MSC specification, behaviours of process P_i are captured by the projection of the language $L(\mathcal{M})$ on P_i, denoted as $L(\mathcal{M})|P_i$, namely the process language.*

From the tester's point of view, $w|P_i$ thus represents the sequence of event labels observed on P_i. For the observation set, we can generalise this in the following way: given set $o \subseteq \{P_1, \ldots, P_n\}$ we let $w|_o$ denote the tuple of projections of w on processes whose labels are in o. It is worth noting that process languages are regular if $\mathcal{M}$ contains only choice and loop structures and may be non-regular if $\mathcal{M}$ contains parallel composition. Where process languages are regular, they can be represented by finite automata, namely *process automata*. The definition of process automata is given as follows.

Definition 2.5. ***(Process automata)*** *A process automaton A_i is a finite automaton that accepts a process language $L(\mathcal{M})|P_i$ where the MSC specification $\mathcal{M}$ contains no parallel composition.*

From an HMSC specification $\mathcal{H}$ with no parallel composition, a process automaton A_i can be generated by projecting $\mathcal{H}$ on process P_i. The projection of $\mathcal{H}$ is basically a relabelling process: using the event labels on P_i of a bMSC node to label the corresponding transitions. The generated automaton has a similar structure to the HMSC, but the transitions of the automaton are labelled by event labels.

3. ORACLES FOR TESTING FROM MSCS

Having run a test we have a set of observations and need to check these observations against the specification: this is the Oracle Problem. Before defining the Oracle Problem for HMSCs, we will describe a general framework that allows us to capture alternative testing scenarios that differ in the observational power of the environment (testers).

Previous work has described a general architecture for testing from MSCs and three scenarios, or notions of observability: local observability, tester observability, and global observability [15]. For testing from MSCs, testers simulate the users' behaviour and check the conformance by comparing the observation of the SUT with behaviour described in the MSC specification. In order to distinguish the behaviour of testers and the SUT, we partition $\mathcal{P}$ into $\mathcal{P}_u$ and $\mathcal{P}_s$: the first represents the set of user processes and the second represents the set of system processes. Under local observability there is an independent tester at each user process in $\mathcal{P}_u$ and the tester at process $P_i \in \mathcal{P}_u$ observes the sequence of events at P_i and compares this sequence with those it might observe according to the specification. In contrast, under tester observability the sequences of events at a process in $\mathcal{P}_u$ is logged, the logs are brought together, and this set of sequences (local traces) is compared with the specification. Finally, under global observability the projections observed at each process in $\mathcal{P}$ are logged and brought together and this information is sufficient to define a bMSC. Clearly, we can order these scenarios in terms of their ability to distinguish an implementation from a specification and under this ordering we have that global observability is the strongest and local observability the weakest [15].

We can generalise the above scenarios in the following way. Given set $\mathcal{P}$ of processes we will let any subset of the power set of $\mathcal{P}$ be called an *observation set*. An observation set $\mathcal{O} = \{o_1, \ldots, o_k\}$ will represent a situation in which each o_i ($1 \leq i \leq k$, $o_i \subseteq \mathcal{P}$) represents a tester that can observe the processes whose labels are in o_i. The tester corresponding to o_i observes the sequences of events on processes in o_i and compares this set of local traces to the specification. It is straightforward to see that the three scenarios described above can be represented in this manner: for local observability we let $\mathcal{O} = \{\{P_i\}|P_i \in \mathcal{P}_u\}$; for tester observability and global observability, as different testers have the same observability we let $\mathcal{O} = \{\{\mathcal{P}_u\}\}$ or $\{\{\mathcal{P}\}\}$, respectively. We call processes in subsets of $\mathcal{O}$ *observable processes*. We can now say what it means for a word w to be acceptable given MSC specification $\mathcal{M}$ and observation set $\mathcal{O}$.

Definition 3.1. ***(Oracle Problem of MSC specification)*** *Given MSC specification $\mathcal{M}$ and observability set $\mathcal{O}$, a word w is* acceptable *if for all $o \in \mathcal{O}$ there exists a word $w_o \in L(\mathcal{M})$ such that $w|_o = w_o|_o$.*

Given MSC specification $\mathcal{M}$ and observation set $\mathcal{O}$, the Oracle Problem is to determine whether a word w is acceptable. We now explore the complexity of this problem for different features of HMSCs.

4. COMPLEXITY WITHOUT PARALLEL COMPOSITION

This section examines the complexity of the Oracle Problem without considering parallel compositions; parallel composition is considered in Section 5. First, we consider the Oracle Problem with local observability. We then examine the two other types of observability (tester and global). In this we discussing two simpler

[4]We also use $\mathcal{H}$ to refer an MSC specification as most MSC specification discussed in this paper are organised by HMSCs.

cases: HMSCs with only choices and with only loops. We then adapt previous results [31, 33] to show that there are polynomial algorithms to solve the Oracle Problem for HMSCs with a bound on the number of processes and also for safely realisable HMSCs.

4.1. Complexity with local observability

In this section we briefly discuss the case where $\mathcal{O}$ is a set of singletons, which corresponds to local observability. We show that for MSC specifications with just choices and loop structures the Oracle Problem can be efficiently solved by Algorithm 1. The complexity of Algorithm 1 is given in the following theorem.

```
   Input: $\mathcal{H}$, $w$, $\mathcal{O}$
   /*$\mathcal{H}$ is the input HMSC without
     parallel compositions, $w$ is an
     observed word and $\mathcal{O}$ is the
     observation set                        */
   Output: True/False
   /*The algorithm returns True if $w$ is
     acceptable, otherwise False is
     returned                               */
1  foreach $o_i \in \mathcal{O}$ with $o_i = \{P_i\}$ do
2      $A_i$ = project($\mathcal{H}$, $P_i$) /*Generate
         process automaton $A_i$ by
         projecting $\mathcal{H}$ to $P_i$ which is the
         only item in $o_i$                 */
3      $w_i = w|P_i$ /*Generate projection on
         $P_i$ from $w$                     */
4      if reject($A_i$, $w_i$) /*Check if $A_i$
         rejects $w_i$                      */
5      then
6          return False
7      end
8  end
9  return True
     Algorithm 1: checkLocalHMSC
```

Theorem 4.1. *Under local observability, the Oracle Problem for HMSC $\mathcal{H}$ that uses no parallel compositions can be solved in time $O(|\mathcal{H}||w|)$, where $|\mathcal{H}|$ is the size of the HMSC, $|w|$ is the length of w.*

Proof. The first step of the algorithm is to generate process automata and this can be achieve in linear time. The second step, which checks whether $w|P_i$ is a word of A_i can be solved in time $O(|\mathcal{H}_i||w|P_i|)$ where $|\mathcal{H}_i|$ is the size of the projection of $\mathcal{H}$ onto the process P_i and $|w|P_i|$ is the length of the projection of w on process P_i. As a result, the overall algorithm operates in time $O(\sum_{i=1}^{n}|\mathcal{H}_i||w|P_i|) = O(|\mathcal{H}||w|)$. □

Consider now the HMSC given in Figure 2. There is a loop around bMSC M_1 in which processes U_1 and U_2 are simulated by testers with local observability. The algorithm accepts an observation $\{\{?a?a\}, \{?b\}\}$ as

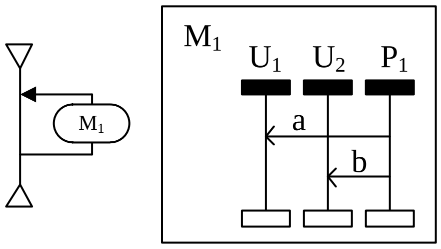

FIGURE 2. Issue of the local observability

$?a?a$ and $?b$ are both valid behaviours according to the corresponding process languages. However, no scenario described by the HMSC generates such an observation. This illustrates how weak local observability can be and why we are therefore also interested in the stronger notions of observability.

4.2. Complexity with stronger observability

With tester or global observability, testers can observe events on the same set of processes $\mathcal{P}_u$ or $\mathcal{P}$, respectively. It is straightforward to see that the Oracle Problem can be solved in polynomial time if $\mathcal{M}$ is a single bMSC and so we explore what happens if we add loops and choice in HMSCs. We leave the consideration of parallel composition to Section 5, since it transpires that this feature can make the Oracle Problem NP-complete even if we are using local observability.

4.2.1. HMSCs with choice

Previous work has considered the membership problem for HMSCs, where we are allowed to have choices and loops but not parallel composition. We therefore start by considering choices and loops. It has been shown that the membership problem for HMSCs is NP-complete, even if the HMSC is acyclic [33]. This extends immediately to the Oracle Problem if we set $\mathcal{P}_u = \mathcal{P}$ and $\mathcal{O} = \{\mathcal{P}\}$; if all events are observed. However, we can show that the Oracle Problem for HMSCs is NP-complete even without the restriction that $\mathcal{P}_u = \mathcal{P}$ and we allow only sequencing and choice. This still holds if we either require that no user process sends a message or that no user process receives a message. We will prove this by reducing the one-in-three SAT problem, which is NP-complete [34], to the Oracle Problem for MSCs.

Definition 4.1. ***(One-in-three SAT problem)*** *Let us suppose that $z_1, \ldots, z_v$ are boolean variables and $C_1, \ldots, C_c$ are sets of three literals, where a literal is either a variable or its negation. The one-in-three SAT problem is to decide whether there is an assignment to the variables such that each C_j, $1 < j < c$, contains exactly one true literal.*

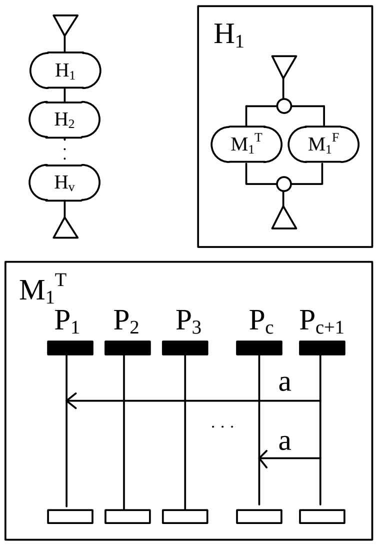

FIGURE 3. HMSC constructed with choices

Theorem 4.2. *The following problem is NP-complete: given a word w, observation set $\mathcal{O}$ and HMSC $\mathcal{H}$ that uses only choices and sequencing, is w acceptable for $\mathcal{H}$. Further, this still holds if we restrict $\mathcal{H}$ so that either no messages are sent by user processes or so that no messages are received by user processes.*

Proof. We first show that the problem is in NP. Given word w, a non-deterministic Turing Machine can guess a word of $\mathcal{H}$ with the same length as w and check whether it is observationally equivalent to w (it has the same set of projections onto the individual processes). We can guess such a word in polynomial time and the process of checking equivalence to w also takes polynomial time since we just generate the local projections and compare these. Thus, a non-deterministic Turing machine can solve this problem in polynomial time and so the problem is in NP as required.

It is now sufficient to prove that the problem is NP-hard. We will assume that an instance of the one-in-three SAT problem has been given with variable set $z_1, \ldots, z_v$ and sets $C_1, \ldots, C_c$ in which each C_j contains three literals and will construct a corresponding instance of the Oracle Problem for an acyclic HMSC $\mathcal{H}$ and a particular observation w. We will construct $\mathcal{H}$ and w so that w is allowed by $\mathcal{H}$ if and only if there is a solution to this instance of the one-in-three SAT problem.

We will define an HMSC as shown in Figure 3 in which there are $c+1$ processes; the processes with labels $1, \ldots, c$ will be user processes and the process with label $c+1$ will be the unique system process. We will also let $\mathcal{O} = \{\{1, \ldots, c\}\}$. We will initially construct $\mathcal{H}$ so that no messages are sent by user processes.

The HMSC $\mathcal{H}$ will be formed of a sequence $H_1, \ldots, H_v$ of HMSCs where H_i will represent a choice for the value of variable z_i. The ith choice, $1 \leq i \leq v$, will have two cases and so H_i contains two bMSCs M_i^T and M_i^F. In M_i^T there is a message a sent from system process $c+1$ to every process j such that C_j contains the literal z_i. In M_i^F there is a message a sent from system process $c+1$ to every process j such that C_j contains the literal $\neg z_i$. These model the two possible values for z_i.

Consider a bMSC in $\mathcal{H}$. Such a bMSC M' is of the form $M_1^{b_1} \ldots M_v^{b_v}$ and represents boolean variable z_i taking on the value b_i, $1 \leq i \leq v$. In addition, process j receives a message in $M_i^{b_i}$ if and only if one of the following holds:

1. b_i is true (T) and the clause C_j contains the literal z_i; or
2. b_i is false (F) and the clause C_j contains the literal $\neg z_i$.

As a result, the number of messages received by process j in M', $1 \leq j \leq c$, is equal to the number of literals in C_j that are true when we have the valuation in which $z_i = b_i$, $1 \leq i \leq v$.

Now consider the word w in which each process in $\mathcal{P}_u$ observes the reception of *exactly* one message. Then w is acceptable given $\mathcal{H}$ and $\mathcal{O}$ if and only if there is a bMSC $M' = M_1^{b_1} \ldots M_v^{b_v}$ such that w is an observation allowed by M'. In addition, since w has each process j receiving exactly one message, we have seen that this is the case if and only if the valuation in which $z_i = b_i$, $1 \leq i \leq v$, is such that each C_j contains exactly one literal that is true. Thus, w is allowed by $\mathcal{H}$ if and only if there is a solution to this instance of the one-in-three SAT problem. Thus, if we can solve this type of Oracle Problem for HMSCs in polynomial time then we can also solve the one-in-three SAT problem in polynomial time. Since we can construct $\mathcal{H}$ in polynomial time, the result follows from the one-in-three SAT problem being NP-complete.

To show that the result also holds if we require that no messages are received by user processes we simply reverse the direction of messages. □

4.2.2. HMSCs with loops

If we have an HMSC that is in the form of a single loop that contains a sequence then the Oracle Problem can be solved in polynomial time: given $o \in \mathcal{O}$ we look at the observation on one process in o in which the loop has a non-empty word and we can then determine the number of iterations the loop must have taken. We then check the other processes in o.

Let us consider an HMSC with a single loop $\mathcal{H}$ with n observable processes. We can generalise the above as shown in Figure 4: one bMSC M_1 before the loop structure, M_2 is in the loop structure and M_3 is after the loop structure. Let w_i^j represent the sequence of event labels of M_j on P_i, where $j \in 1, 2, 3$. The algorithm that checks whether observation w is acceptable is given in

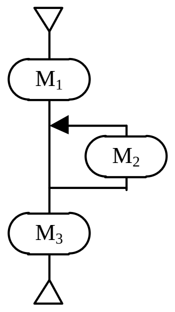

FIGURE 4. HMSC with a single loop

Algorithm 2. In the algorithm, given words u, v and w, function *isSuffix(v,w)* returns *true* if and only if v is a suffix of w; function *isPrefix(v,w)* returns *true* if and only if v is a prefix of w; function *trimPrefixSuffix(w, u, v)* returns a word formed by removing prefix u and suffix v from w; function *isRepetition(v,w)* returns *true* if and only if $v = w^n$ where $n >= 1$; *repetition(v,w)* returns integer n if $v = w^n$. Algorithm 2 compares the number of iterations of the loop for all processes. This is because there is a possibility that different processes take varying numbers of iterations. In this case, the observation should be rejected. The complexity of Algorithm 2 is given in the following theorem.

Theorem 4.3. *Under tester observability with n observable processes, the Oracle Problem for word w and HMSC $\mathcal{H}$ that contains a single loop and does not use choices or parallel composition can be solved in time $O(n|\mathcal{H}||w|)$.*

Proof. There are n iterations of the outer loop. The iteration for P_i starts by removing w_i^1 and w_i^3 from $w|P_i$, if this is possible, and so takes $O(|w|)$ time. The loop count is then determined by simulating $\mathcal{H}_i$ with w_i' and this takes $O(|\mathcal{H}_i||w_i'|)$ time. Thus, the overall complexity is $O(n|\mathcal{H}||w|)$ as required. □

In general, however, the Oracle Problem is NP-complete for HMSCs in which we have loops and sequencing, the following adapting the proof of Theorem 4.2. Essentially, we can use loops to simulate choices through the cases where the body of the loop is not executed and where it is executed exactly once.

Theorem 4.4. *The following problem is NP-complete: given a word w, observation set $\mathcal{O}$ and HMSC $\mathcal{H}$ that uses only loops and sequencing, is w acceptable for $\mathcal{H}$. Further, this still holds if we restrict $\mathcal{H}$ so that either no messages are sent by user processes or so that no messages are received by user processes.*

Proof. The proof that the problem is in NP is equivalent to that in the proof of Theorem 4.2. We therefore need to prove that the problem is NP-hard and again

```
Input: $\mathcal{H}$, $w$, $o$
/*$\mathcal{H}$ is the input HMSC with a single
  loop, $w$ is an observed word and $o$
  is the only subset in $\mathcal{O}$ where
  $o = \mathcal{P}_u$ for tester observability;
  $o = \mathcal{P}$ for global observability   */
Output: True/False
1  $lastLC = 0$ /*Initialise the last loop
     counter                                  */
2  foreach $P_i \in o$ do
3      $w_i = w|P_i$ /*Generate projection on
         $P_i$ from $w$                       */
4      if not $isPrefix(w_i^1, w_i)$ then
5          return False
6      end
7      if not $isSuffix(w_i^3, w_i)$ then
8          return False
9      end
10     $w_i' = trimPrefixSuffix(w_i, w_i^1, w_i^3)$
11     if not $isRepetition(w_i', w_i^2)$ then
12         return False
13     end
14     if $w_i^2 = \epsilon$ then
15         continue
16     end
17     $lc = repetition(w_i', w_i^2)$
18     if (lastLC $\neq$ lc) $\wedge$ (lastLC $\neq$ 0) then
19         return False
20     end
21     $lastLC = lc$
22 end
23 return True
```
Algorithm 2: checkSingleLoopHMSC

we assume that an instance of the one-in-three SAT problem has been given with variable set $z_1, \ldots, z_v$ and sets $C_1, \ldots, C_c$ of three literals and will construct a corresponding instance of the Oracle Problem for an HMSC that makes no use of choice. We will define an HMSC in which there are $c+2$ processes; the processes with labels $0, 1, \ldots, c$ will be user processes and the process with label $c+1$ will be the system process. We also let $\mathcal{O} = \{\{0, 1, \ldots, c\}\}$ and will initially construct $\mathcal{H}$ so that no messages are sent by user processes.

The HMSC $\mathcal{H}$ will be formed of a sequence of v HMSCs as shown in Figure 5. There are two loops for each variable in each HMSC. The first loop of H_i, $1 \leq i \leq v$, will have an MSC M_i^T in which there is a message with label a sent from system process $c+1$ to every process j such that C_j contains the literal z_i and also a message with label i sent from $c+1$ to process 0. In the second loop (M_i^F), there is a message a sent from system process $c+1$ to every process j such that C_j contains the literal $\neg z_i$ and also a message with label i sent from $c+1$ to process 0. These loops model the two possible values for z_i.

Now consider the word w in which each process in

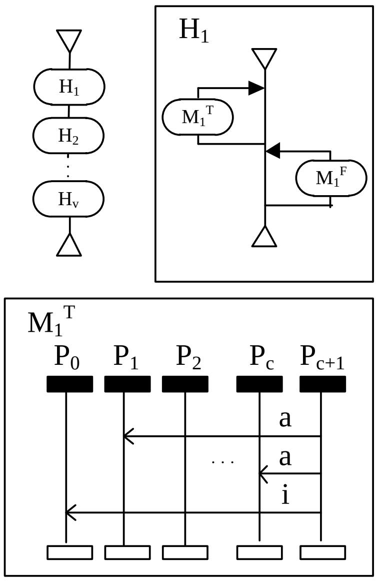

FIGURE 5. HMSC constructed with loops

$\{1, \ldots, c\}$ receives a and process 0 receives $1, \ldots, v$. The observation on process 0 ensures that we must have entered the body of a loop corresponding to a value for each z_i, $1 \leq i \leq v$, and have done this exactly once. Then w is acceptable given $\mathcal{H}$ and $\mathcal{O}$ if and only if there is an assignment to the boolean variables $z_1, \ldots, z_v$ such that each C_j contains exactly one true literal. Since we can construct $\mathcal{H}$ in polynomial time, the result follows from the one-in-three SAT problem being NP-complete.

To show that the result also holds if we require that no messages are received by user processes, we simply reverse the direction of messages. □

4.3. Complexity with other restrictions

In this section, we discuss two restrictions on HMSCs which reduce the complexity of the Oracle Problem for testing from HMSCs. One is where there is a bound on the number of processes. The other is that either the HMSC specification $\mathcal{H}$ is safely realisable [5] or $\mathcal{H}$ is not safely realisable but the implied scenarios of $\mathcal{H}$ are acceptable in testing.

Alur et al. [33] showed that the membership problem can be solved in polynomial time if we place a bound on the number of processes. We can adapt this to show that when we have choices and loops the Oracle Problem can be solved in polynomial time if we place a bound on the size of $\mathcal{O}$.

THEOREM 4.5. *Given an HMSC $\mathcal{H}$, observation w and a bound k on the size of sets in $\mathcal{O}$, the Oracle Problem can be solved in time $O(|\mathcal{H}||w|^k)$.*

[5]The concept of *safely realisable* is introduced later in this section.

Proof. This theorem follows the membership results given in Theorem 5 in Alur et al. [33]. □

This result is especially useful with tester observability as there often are a limited number of testers in a distributed test environment. The corresponding solution is given in Algorithm 3 which is a recursive procedure that is run for each $o \in \mathcal{O}$. Basically, it searches HMSC $\mathcal{H}$ by matching each next MSC node according to observation w. The extreme case is that the search process has to traverse $|\mathcal{H}||w|^k$ nodes to give the decision.

```
Input: $\mathcal{H}$, $w$, $o$, $c$
/*$\mathcal{H}$ is the input HMSC, $w$ is an
  observed word, $o$ is the set in $\mathcal{O}$
  containing $k$ processes and $c$ is
  current node in $\mathcal{H}$ which initially
  is the start node                     */
Output: True/False
tc = c /*Backup $c$                       */
while $w \neq \epsilon$ do
    foreach $M \in \mathcal{H}$ /*$M$ is next
      possible MSC node in $\mathcal{H}$        */
    do
        if $\exists w' \in L(M)|_o$ is head of $w$ then
            $w = w - w'$ /*Remove the head
              of $w$                      */
            $c = M$
            call checkBoundedHMSC ($\mathcal{H}$, $w$,
            $o$, $c$)
        end
    end
    if $c = tc$ /*Cannot find a successive
      node                              */
    then
        break
    end
end
if $w = \epsilon \wedge isTermination(c)$ /*Check if $c$
  is a termination node and $w$ is an
  empty string                          */
then
    return True
end
```
Algorithm 3: checkBoundedHMSC

The second restriction is based on research on implied scenarios in HMSC specification. The definition of implied scenario can be found in [31].

DEFINITION 4.2. ***(Implied Scenarios)*** *w represents an* implied scenario *of MSC specification $\mathcal{M}$ if w is a well-formed word and for each $w|P_i$ $i \in [n]$, a word $v \in pref(L(\mathcal{M}))$ exists such that $w|P_i = v|P_i$, but $w \notin pref(L(\mathcal{M}))$.*

According to [31, 35], the synthesised model from an HMSC specification may exhibit additional behaviours which are not described by the original specification even though each process follows its local specification; these are the implied scenarios. For example, consider the MSC specification described by H_1 in Figure 6. M_3 is an implied scenario of H_1 since the behaviours of all processes follows the specification. The behaviours of P_1 and P_2 conform to the left branch of H_1 and the behaviours of P_3 and P_4 conform to the right branch of H_1. However, M_3 is not described by H_1.

An HMSC specification $\mathcal{M}$ is said to be *safely realisable* if and only if there exists a synthesised model whose behaviour contains no implied scenarios and also no deadlocks [31]. The decision problem regarding whether an HMSC is safely realisable is generally undecidable but it is in EXPSPACE and is PSPACE-hard if the number of pending messages is limited to be a finite number since all the loops of the HMSC are roughly synchronised [33].

The Oracle Problem and the safely realisable decision problem are different. However, safely realisable HMSC specifications have the following property: a word w is in the language of an MSC specification $\mathcal{M}$ if and only if for every process P_i we have that the projection of w on P_i is a word of the process language of $\mathcal{M}$ at P_i. Therefore, to solve the Oracle Problem with the second restriction, for each $o \in \mathcal{O}$ and $P_i \in o$ it is sufficient to check that the projection of the word on P_i is in the corresponding process language. The algorithm is given in Algorithm 4 which is similar to Algorithm 1. Note that Algorithm 4 is applied with each $o \in \mathcal{O}$. Both algorithms work on process behaviours and the difference is the observabilities of testers.

```
Input: $\mathcal{H}$, $w$, $o$
/*$\mathcal{H}$ is the input HMSC which is
  safely realisable, $w$ is an
  observed word and $o$ is the set in
  $\mathcal{O}$                                  */
Output: True/False
1 foreach $P_i \in o$ do
2     $A_i$ = project($\mathcal{H}$, $P_i$) /*Generate
        process automaton $A_i$ by
        projecting $\mathcal{H}$                 */
3     $w_i = w|P_i$ /*Generate projection on
        $P_i$ from $w$                          */
4     if reject($A_i$, $w_i$) /*Check if $A_i$
        rejects $w_i$                           */
5     then
6         return False
7     end
8 end
9 return True
```
Algorithm 4: checkSafelyRealisableHMSC

Theorem 4.6. *Given a safely realisable HMSC $\mathcal{H}$, observation w, the Oracle Problem can be solved in time $O(|\mathcal{H}||w|)$ where $|\mathcal{H}|$ is the size of the HMSC, $|w|$ is the length of w.*

Proof. This theorem can be proven as follows. There are two steps in the algorithm. First, building an automaton representing a process language of HMSC $\mathcal{H}$ without parallel compositions can be done in linear time. Second, checking whether the projection of observed word $w|P_i$ is in the corresponding process language can then be done in time $O(|\mathcal{H}_i||w|P_i|)$ where $|\mathcal{H}_i|$ is the size of the projection of $\mathcal{H}$ onto the process P_i. Thus the total time is of $O(|\mathcal{H}||w|)$ □

4.4. Summary

Table 1 summarises the complexity results given in this section. A cell of the table shows the complexity of the class of Oracle Problem for HMSCs with particular features given by the column and row headers. Different columns represent HMSCs with different types of constructs, for example column 3 represents HMSCs with only choice structures. Rows of the table introduce other configurations such as the type of observability (Obser.) or other restrictions (Other restr.) such as row 3 representing the MSC being safely realisable and row 4 representing there being a bound on the size of the observation set. We use "tester+" to represent both tester and global observability as there is no difference, in terms of complexity results, between these two types of observability with each observation set containing multiple processes. For local observability there is only one row because the results do not change if we assume that an MSC is safely realisability or there is a bound on the number of processes.

For the first three rows of the table, we have proved that the complexity of these instances of the Oracle Problem for HMSCs with choice and loop structures (the last column) can be solved in polynomial time. Thus, this result holds immediately for HMSCs with only choice, single loop or loop structures.

5. PARALLEL COMPOSITION

From Section 4.1, we know that the Oracle Problem with local observability and no use of the parallel feature can be solved in polynomial time. We now explore the effect of the parallel composition under local observability.

5.1. HMSCs with choices or loops

We use a similar approach to that given in Section 4 to discuss the complexity of the Oracle Problem for HMSCs with parallel compositions plus choices or loops.

Theorem 5.1. *Under local observability, if we allow specifications to contain the choice and parallel features then the Oracle Problem is NP-complete. This is the case even if we restrict attention to MSCs with only two processes.*

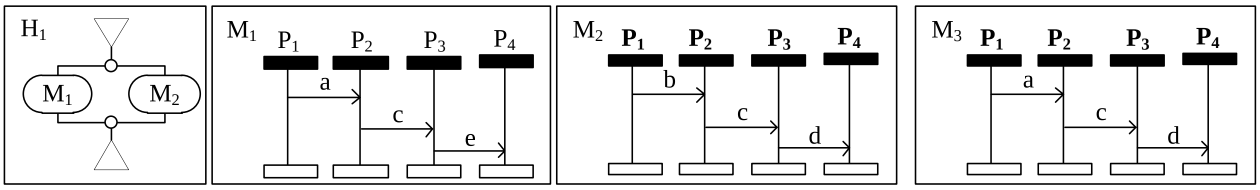

FIGURE 6. Example of implied scenarios

TABLE 1. Complexities of Oracle Problem without parallel compositions

Obser.	Other restr.	Choice	Single Loop	Loop	Choice+Loop
Local	None	P	P	P	P[a]
Tester+	R	P	P	P	P[b]
	BO	P	P	P	P[c]
	None	NP-complete[d]	P[e]	NP-complete[f]	NP-complete

[a]Theorem 4.1
[b]Theorem 4.6
[c]Theorem 4.5
[d]Theorem 4.2
[e]Theorem 4.3
[f]Theorem 4.4

Proof. The proof that the problem is in NP is equivalent to that in the proof of Theorem 4.2. To prove that the problem is NP-hard we again use the one-in-three SAT problem; we assume that an instance of the one-in-three SAT problem has been given with variable set $z_1, \ldots, z_v$ and sets $C_1, \ldots, C_c$ of three literals and will construct a corresponding instance of the Oracle Problem for an HMSC in which we use choice structures and parallel compositions.

We will form an HMSC as shown in Figure 7 in which there are two processes P_1, P_2 and in which P_1 is the system process, P_2 is the user process and $\mathcal{O} = \{\{P_2\}\}$. We will also use a set $\{m_1, \ldots, m_c\}$ of message labels.

We will have v separate choices in parallel. The ith choice, $1 \leq i \leq v$, will have two options.

1. Under one option, corresponding to the case where z_i is true, there is a sequence of messages from P_1 to P_2, with there being a message with label m_j if and only if C_j contains literal z_i. These messages are received in the order implied by the subscripts: if both m_r and m_s are sent and $r < s$ then m_r is received before m_s.
2. Under the other option, corresponding to the case where z_i is false, there is a sequence of messages from P_1 to P_2, with there being a message with label m_j if and only if C_j contains literal $\neg z_i$. These messages are received in the order implied by the subscripts: if both m_r and m_s are sent and $r < s$ then m_r is received before m_s.

We now ask whether user process P_2 can observe the sequence of messages with labels $m_1, \ldots, m_c$ in that order. Clearly, this is allowed by $\mathcal{H}$ if and only if there is an assignment to the boolean variables (a set of choices) under which each C_j contains exactly one literal that is true. The result now follows from the one-in-three SAT problem being NP-complete and the fact that we can form $\mathcal{H}$ in polynomial time. □

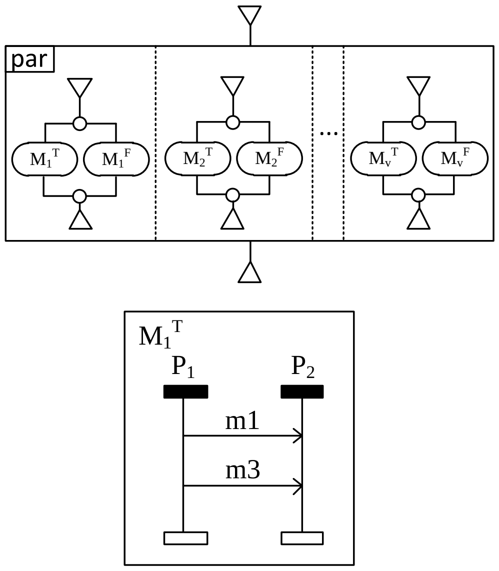

FIGURE 7. HMSC constructed with parallel compositions

As with the proof of Theorem 4.4, it is possible to use loops to simulate choices. It is therefore straightforward to prove the following.

Theorem 5.2. *Under local observability, if we allow specifications to contain the loop and parallel features then the Oracle Problem is NP-complete. This is the case even if we restrict attention to MSCs with only two processes.*

5.2. Complexity with other restrictions

Since the proofs of Theorem 5.1 and 5.2 used a specification with only two processes, the problem is still NP-complete if we place bounds on the number of processes. However, if we place bounds on the number of components we have in the parallel composition then the problem becomes polynomial with local observability.

Theorem 5.3. *Let us suppose that specification $\mathcal{H}$ with n observable processes is in the form of a parallel composition of at most k HMSCs that do not contain the parallel feature. Under local observability, the Oracle Problem with word w can be solved in $O(n|\mathcal{H}|^k|w|)$ time.*

Proof. Let $\mathcal{H}$ denote the specification and assume that $\mathcal{H}$ is the parallel composition of $H_1, \ldots, H_k$. For tester process P_i and H_j, $1 \leq j \leq k$, let A_{ij} be the finite automaton formed by taking the projection of H_j on process P_i. Then we define a finite automaton A_i in the following way. A state of A_i is of the form $(s_1, \ldots, s_k)$ where s_l is a state of A_{il} for $1 \leq l \leq k$. If A_{ij} has a transition from s_j to s'_j with label a then we include in A_i transitions of the following form: there is a transition with label a from $(s_1, \ldots, s_{j-1}, s_j, s_{j+1}, \ldots, s_k)$ to $(s_1, \ldots, s_{j-1}, s'_j, s_{j+1}, \ldots, s_k)$. A state $(s_1, \ldots, s_k)$ of A_i is a final state if and only if s_j is a final state of A_{ij} for all $1 \leq j \leq k$. Then $L(A_i)$ is the set of interleavings of the words in $L(A_{ij})$ and this is exactly the set of words that are allowed by $\mathcal{H}$ at P_i. Thus, given word w we can solve the Oracle Problem by constructing the A_i and solving the membership problem for $w|P_i$ and A_i for each tester process P_i. Now observe that each A_{ij} can be constructed in linear time and has size of $O(|\mathcal{H}|)$ and so the A_i have at most $|\mathcal{H}|^k$ states. The result now follows from there being $O(n)$ membership problems, one per process, and the fact that a membership problem with a word of length $O(|w|)$ and finite automaton with $O(|\mathcal{H}|^k)$ states can be solved in $O(|\mathcal{H}|^k|w|)$ time. ☐

The algorithm developed based on Theorem 5.3 is given in Algorithm 5. The first loop structure is used to generate the set of process automata A_{ij} in parallel. In the second loop, line 7 generates the set of process automaton A_i by taking the product of the A_{ij}. It then checks whether the projection of the observation on process P_i is accepted by the corresponding A_i.

Now we show that the restriction that requires HMSCs being safely realisable does not reduce the complexity.

```
   Input: $\mathcal{H}$, $w$, $\mathcal{O} = \{\{P_1\}, \ldots, \{P_n\}\}$
   /*$\mathcal{H}$ is the only parallel composition
     and contains $k$ components          */
   Output: True/False
1  foreach $H_j \in \mathcal{H}$ do
2      foreach $P_i$ do
3          $A_{ij}$ = project($H_j$, $P_i$) /*Generate
             process automaton of $H_j$ for
             $P_i$                            */
4      end
5  end
6  foreach $P_i$ do
7      $A_i = \prod_{j=1}^{k} A_{ij}$ /*Product automata to
         generate process automata for
         the parallel composition       */
8      $w_i = w|P_i$ /*Generate projection on
         $P_i$ from $w$                    */
9      if reject($A_i$, $w_i$) /*Check if $A_i$
         rejects $w_i$                    */
10     then
11         return False;
12     end
13 end
14 return True
```
Algorithm 5: checkBoundedParallelHMSC

Theorem 5.4. *Let us suppose that specification $\mathcal{H}$ is safely realisable and contains the choice and parallel features. Under local observability, the Oracle Problem with $\mathcal{H}$ is NP-complete.*

Proof. Consider the HMSC used in the proof of Theorem 5.1. This HMSC does not contain non-local choices and so is safely realisable according to Proposition 23 in [36]. The result therefore follows. ☐

We now consider another restriction on HMSCs in which no event label appears in more than one parallel component of an HMSC. Then if we consider a word, for each observed label we can uniquely determine the parallel component that it must have come from. It is then possible to solve the Oracle Problem for local observability in polynomial time since we just compare the projections of observation to the HMSC $\mathcal{H}$. Let $\mathcal{H}$ be the parallel composition of H_i for $1 \leq i \leq k$ and let $\mathcal{A}^{\mathcal{H}}$ be the set of event labels in $\mathcal{H}$. The algorithm that solves the corresponding Oracle Problem is given in Algorithm 6.

The complexity of Algorithm 6 is stated in the following theorem.

Theorem 5.5. *Let us suppose that specification $\mathcal{H}$ with n observable processes is in the form of a parallel composition of HMSCs $H_1, \ldots, H_k$ that do not contain the parallel feature. Further let us suppose that the H_j, $1 \leq j \leq k$, have disjoint sets of event labels. Under local observability, the Oracle Problem with word w can be solved in $O(n|\mathcal{H}||w|)$ time.*

```
Input: $\mathcal{H}$, $w$, $\mathcal{O} = \{\{P_1\}, \ldots, \{P_n\}\}$
/*$\mathcal{H}$ is the parallel composition of $H_1, \ldots, H_k$ */
Output: True/False
1  foreach $1 \leq j \leq k$ do
2      $w^j = w|\mathcal{A}^{H_j}$ Project $w$ onto the set of event labels of $H_j$
3  end
4  foreach $P_i$ do
5      $w_i^j = w^j|P_i$
6      $A_i^j$ = project($H_j$, $P_i$) /*Process automaton $A_i^j$ is generated by projecting $H_j$ on $P_i$ */
7      if reject($A_i^j$, $w_i^j$) then
8          return False
9      end
10 end
11 return True
```
Algorithm 6: checkDisjointParallelHMSC

Proof. Let us suppose that $\mathcal{H}$ is the parallel composition of HMSCs $H_1, \ldots, H_k$. Given word w and H_j we can form the word w^j that is the projection of w on the set of event labels of H_j and this can be achieved in linear time. It is then sufficient to solve the Oracle Problem for each w_i and H_j. The Algorithm thus repeats Algorithm 1 for each parallel component and so the result follows from Theorem 4.1 and the fact that $|\mathcal{H}|$ is of $O(\sum_{j=1}^{k} |H_j|)$. □

5.3. Summary

Table 2 summarises the complexity results for HMSCs with parallel composition. For local observability, it is shown that HMSCs with choice or loop structures lead to NP-complete complexity. Therefore, for HMSCs with both structures, it is clear the Oracle Problem is NP-complete. As the bound on the number of parallel components reduces the complexity of the Oracle Problem to polynomial time for HMSCs with both choice and loop structures, the corresponding problem for HMSCs with choice or loop structures can also be solved in polynomial time.

In terms of tester+ observability, the Oracle Problem is NP-complete even under the restriction BP. This is because we have shown that for HMSCs with choice or loop structures but no parallel composition the problem is NP-complete.

6. CONCLUSIONS

Message Sequence Charts (MSCs) forms a popular language in which scenario-based specifications and models can be written. Since MSCs are widely used in industry, and are suitable for describing distributed systems, there has been significant interest in automating aspects of testing from MSCs. The focus of previous work has largely been on automated test generation but this paper concerns a complementary issue: having an automated test Oracle based on an MSC.

Before devising a test Oracle one has to decide upon the observational power of the environment (or tester), since this determines the nature of the observations we are considering. This paper defined a general form of the Oracle Problem and considered three cases of this: local observability; tester observability; and global observability. It then considered the complexity of the Oracle Problem for these forms of observability with HMSCs that use particular features.

We first considered the case where parallel composition is not used. It transpired that the Oracle Problem is simpler with local observability. Specifically, this can be solved in low-order polynomial time for HMSCs that contain loops and/or choices. In contrast, for tester and global observability, the problem is NP-complete when we have either loops or choices. However, it can be solved in low-order polynomial time if we have only one loop or have an HMSC which is safely realisable. In addition, it was also shown that the complexity can be reduced to polynomial time by putting a bound on the number of processes of HMSCs.

We then allowed parallel composition and found that the Oracle Problem is NP-complete even for local observability. However, for local observability the problem can be solved in polynomial time if we place a bound on the number of parallel components. Under tester and global observability, the Oracle Problem is NP-complete even if we restrict attention to HMSCs in which there are at most two parallel components.

While there has been significant interest in the use of scenario-based models represented as MSCs or Sequence Diagrams (SDs), the results in this paper provide a warning to practitioners. This complements previous results suggesting that the semantics of scenario-based models can be difficult to understand, leading to the potential for subtle mistakes [18]. A tester wishing to base testing on MSCs must consider the semantic issue and the results in this paper also show that the set of features used can have another significant practical effect in that it influences the computational complexity of the Oracle Problem. The results thus provide a warning regarding potential practical issues and define a type of testability under which these issues are less significant. It may well be that testers will wish to restrict attention to sets of features where semantic misunderstandings are less likely and the Oracle Problem can be solved in polynomial time.

Finally, it is worth noting that results in this paper also apply to SDs of UML, which is another popular scenario-based language [37]. It is clear that we can construct SDs modelling the behaviours described in Figures 3, 4, 5 and 7 with the main constructs of SDs: *Lifeline*, *Message*, *Occurrence* and *CombinedFragment* with *InteractionOperators seq*, *opt*, *par* or *loop*. This

TABLE 2. Complexities of Oracle Problem with parallel compositions

Obser.	Other restr.	Paral.+Choic.	Paral+Loop	Paral.+Choic.+Loop
Local	BP	P	P	P[a]
	R	NP-complete [b]	NP-complete	NP-complete
	None	NP-complete [c]	NP-complete[d]	NP-complete
Tester+	BP	NP-complete	NP-complete	NP-complete
	None	NP-complete	NP-complete	NP-complete

[a]Theorem 5.3
[b]Theorem 5.4
[c]Theorem 5.1
[d]Theorem 5.2

means that the key theorems given in the paper (Theorems 4.2, 4.3, 4.4 and 5.1) apply to SDs with these main constructs. Therefore, when testing from SDs, the Oracle Problem for the corresponding test configurations has at least the same complexity.

There are several lines of future work. First, it may be possible to identify additional conditions under which the Oracle Problem can be solved in polynomial time. In addition, while we considered the main features of HMSCs, it would be interesting to investigate the complexity of the Oracle Problem for HMSCs that use other parts of the language such as time constraints and data tags. Finally, there is also scope to develop a model-based testing tool based on MSCs in which the algorithms given in the paper are implemented to solve the Oracle Problem.

REFERENCES

[1] Grieskamp, W., Kicillof, N., Stobie, K., and Braberman, V. A. (2011) Model-based quality assurance of protocol documentation: tools and methodology. *Softw. Test., Verif. Reliab.*, **21**, 55–71.

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

[3] 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.

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

[5] SDL Forum Society (Accessed in 2012). http://www.sdl-forum.org/.

[6] Krüger, I., Grosu, R., Scholz, P., and Broy, M. (1998) From MSCs to statecharts. *Proceedings of International Workshop on Distributed and Parallel Embedded Systems*, Schloss Eringerfeld, Germany, pp. 61–71., Kluwer Academic Publishers Norwell, MA, USA

[7] Whittle, J., Kwan, R., and Saboo, J. (2005) From scenarios to code: An air traffic control case study. *Software and Systems Modeling*, **4**, 71–93.

[8] 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.

[9] Grabowski, J. SDL and MSC Based Test Case Generation – An Overall View of the SAMSTAG Method. Technical report. IAM-94-0005, University of Berne, 1994.

[10] ITU-T (1998). Recommendation X.292 OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications The Tree and Tabular Combined Notation (TTCN).

[11] Grabowski, J., Hogrefe, D., Réthy, G., Schieferdecker, I., Wiles, A., and Willcock, C. (2003) An introduction to the testing and test control notation (TTCN-3). *Computer Networks*, **42**, 375 – 403.

[12] 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.

[13] 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.

[14] 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 Berlin, Heidelberg.

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

[16] 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, Germany, pp. 49–65, Springer Berlin, Heidelberg.

[17] Dan, H. and Hierons, R. M. (2012) Controllability problems in MSC-based testing. *The Computer Journal*, **55**, 1270–1287.

[18] Dan, H. Hierons, R. M., and Counsell, S. (2012) A framework for pathologies of message sequence charts. *Information and Software Technology*, **54**, 1283–1295.

[19] Lee, N. H. and Cha, S. D. (2003) Generating test sequences from a set of MSCs. *Computer Networks*, **42**, 405 – 417. ITU-T System Design Languages (SDL).

[20] Ulrich, A., Alikacem, E.-H., Hallal, H., and Boroday, S. (2010) From scenarios to test implementations via promela. In Petrenko, A., Simo, A., and Maldonado, J. (eds.), *Testing Software and Systems*, Lecture Notes in Computer Science, **6435**, pp. 236–249. Springer Berlin, Heidelberg.

[21] Damm, W. and Harel, D. (2001) LSCs: breathing life into message sequence charts. *Formal Methods in System Design*, **19**, 45–80.

[22] Harel, D. and Marelly, R. (2003) Specifying and executing behavioral requirements: the play-in/play-out approach. *Software and System Modeling*, **2**, 82–107.

[23] Kugler, H., Stern, M., and Hubbard, E. (2007) Testing scenario-based models. In Dwyer, M. and Lopes, A. (eds.), *Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science, **4422**, pp. 306–320, Springer Berlin, Heidelberg.

[24] Wang, Z., Yin, X., Xiang, Y., Zhu, R., Gao, S., Wu, X., Liu, S., Gao, S., Zhou, L., and Li, P. (2009) TTCN-3 based conformance testing of mobile broadcast business management system in 3G networks. In Nunez, M., Baker, P., and Merayo, M. (eds.), *Testing of Software and Communication Systems*, Lecture Notes in Computer Science, **5826**, pp. 163–178, Springer Berlin, Heidelberg.

[25] Roychoudhury, A., Goel, A., and Sengupta, B. (2012) Symbolic message sequence charts. *ACM Trans. Softw. Eng. Methodol.*, **21**, 12:1–12:44.

[26] Briand, L. C. and Labiche, Y. (2002) A UML-based approach to system testing. *Software and Systems Modeling*, **1**, 10–42.

[27] Garousi, V., Briand, L. C., and Labiche, Y. (2006) A quantitative framework for predicting resource usage and load in real-time systems based on UML models. Technical Report SCE-06-05. Carleton University.

[28] Pickin, S., Jard, C., Jeron, T., Jezequel, J.-M., and Le Traon, Y. (2007) Test synthesis from UML Models of distributed software. *Software Engineering, IEEE Transactions on*, **33**, 252 –269.

[29] Bandyopadhyay, A. and Ghosh, S. (2009) Test input generation using UML sequence and state machines models. *Software Testing Verification and Validation, 2009. ICST '09. International Conference on*, april, pp. 121 –130, IEEE Computer Scoiety.

[30] Mussa, M. and Khendek, F. (2012) Towards a model based approach for integration testing. *Proceedings of SDL 2011: Integrating System and Software Modeling*, Toulouse, France, pp. 106–121. Springer Berlin, Heidelberg.

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

[32] Mauw, S. and Reniers, M. A. (1997) High-level message sequence charts. *Proceedings of 8th International SDL Forum*, Evry, France, pp. 291–306, Elsevier, Amsterdam, Netherlands

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

[34] Schaefer, T. J. (1978) The complexity of satisfiability problems. *Proceedings of the tenth annual ACM symposium on Theory of computing*, New York, NY, USA STOC '78, pp. 216–226, ACM, New York, USA.

[35] 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.

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

[37] OMG (2009). OMG UML Superstructure, Version 2.2.