The Complexity of Asynchronous Model Based Testing

Robert M. Hierons[a]

[a] *The School of Information Systems, and Computing Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK*

Abstract

In model based testing (MBT), testing is based on a model M that typically is expressed using a state-based language such as an input output transition system (IOTS). Most approaches to MBT assume that communications between the system under test (SUT) and its environment are synchronous. However, many systems interact with their environment through asynchronous channels and the presence of such channels changes the nature of testing. In this paper we investigate the situation in which the SUT interacts with its environment through asynchronous channels and the problems of producing test cases to reach a state, execute a transition, or to distinguish two states. In addition, we investigate the Oracle Problem. All four problems are explored for both FIFO and non-FIFO channels. It is known that the Oracle Problem can be solved in polynomial time for FIFO channels but we also show that the three test case generation problems can also be solved in polynomial time in the case where the IOTS is observable but the general test generation problems are EXPTIME-hard. For non-FIFO channels we prove that all of the test case generation problems are EXPTIME-hard and the Oracle Problem in NP-hard, even if we restrict attention to deterministic IOTSs.

Keywords: Software Testing; Asynchronous Channels; Model Based Testing; Input Output Transition System.

1. Introduction

It is widely accepted that testing is an important part of the software development process. However, testing is often a manual process and as a result it is expensive and error prone. This has led to interest in model based testing (MBT), in which a model of some aspect of the system under test (SUT) is produced and testing is based on this model [5, 15, 16, 17, 18, 26, 49]. The models used in MBT are usually state-based and MBT tools typically translate the models into either finite state machines (FSMs) or input output transition system (IOTSs). Automated testing then proceeds on the basis of the FSM or IOTS [16].

Many algorithms for generating test cases from an FSM use input sequences or adaptive processes to reach and distinguish states [1, 2, 13, 21, 27, 34, 38] and there has also been interest in such problems when testing from an IOTS [29]. There are also approaches that aim to execute the transitions of the FSM or IOTS. Thus, approaches to produce (parts of) test cases that reach states, execute transitions and distinguish states play an important role in MBT.

FSMs and IOTSs model interaction as being synchronous. However, in practice communication is often asynchronous: interaction with the SUT proceeds through the exchange of messages using (asynchronous) *channels*. Many types of systems, such as web services, communicate with their environment through such channels. The use of asynchronous channels affects testing through introducing latency: the SUT receives inputs after they were sent and the outputs produced by the SUT are observed later than they were produced. For example, if the tester applies input $?i_1$ followed by input $?i_2$ and then observes output $!o_1$ then the output may have been produced by the SUT after the SUT received the first input: it was not observed by the tester until after the tester sent the second input because of the time taken for the output to arrive. Thus, the observed sequence need not be one that was produced by the SUT. In addition, it can be hard to ensure that each input is received by the SUT after a required sequence of inputs and outputs: in the above example the intention may have been for the SUT to receive $?i_2$ before it sent $!o_1$ but we cannot know whether this has been achieved. Thus, most MBT methods cannot be directly applied when the tester and the SUT communicate through asynchronous channels.

It is desirable to extend the work on testing from an FSM or an IOTS to testing through asynchronous channels. One approach is to model the channels and produce an FSM or IOTS that models the composition of the required behaviour of the SUT and the channels [30, 45, 47]. Traditional FSM and IOTS based test methods can then be applied directly. While this provides a general approach, it has the disadvantage of increasing the number of states of the model being analysed. In particular, if we consider unbounded channels then we compose the original model with one or more infinite state models and so obtain an infinite state machine. Even if the channels are bounded, the number of states of the composition of the original model and the model of the channels grows exponentially with the bound. An alternative is to reason about the effect of the channels and this is the approach we take in this paper. In this context, there has been work that has defined alternative implementation relations (notions of correctness) for testing in the presence of asynchronous channels [6, 7, 8, 9] and also research that has explored test case generation problems [39, 28, 29]. There has also been some work that has looked at the situation in which inputs are supplied in a synchronous manner but outputs are asynchronous [36]. Finally, it has been shown that if inputs and outputs are stamped to show the order in which they occurred then the use of asynchronous communications has less effect [31] but this requires the SUT to communicate synchronously with the agent that stamps the events.

MBT test case generation algorithms often aim to test aspects of the SUT associated with parts of the model M being used. For example, one standard test criterion is to execute test cases that reach all states, while another is to execute all transitions [29, 35, 13]. In addition, in testing we will often wish to set up the state in order to execute (check) a transition and then check that the state after the transition is correct. To check the state after a transition we typically use input sequences or strategies (also called adaptive test cases) that distinguish between the expected state and alternatives [3, 27, 32, 32]. However, only recently have such problems been considered for (asynchronous) testing through channels [29]. This recent work [29] investigated the problem of finding input sequences to execute transitions of an IOTS model. The authors distinguish between two cases: the sequence must execute the transition t of interest and the sequence may execute t. The 'may' case is relatively straightforward: any test case that executes a transition t of an IOTS under synchronous testing may also execute it under asynchronous testing. As a result, in this paper we investigate the problem of finding test cases that *must* achieve a given objective. The approach that has been given for finding sequences that must execute a transition t when there are asynchronous channels is to first produce a test case that must execute t under synchronous communications and then check whether it still achieves this under asynchronous communications. While the results of empirical studies were encouraging [29], this paper did not investigate the problem of deciding whether there is a test case that must execute a given transition. An alternative approach adapts a test purpose, produced from the specification IOTS, for use in asynchronous testing but does not directly consider problems such as reaching or distinguishing states [14].

It is known that there are polynomial time algorithms for finding sequences to reach a state of a deterministic finite state machine (DFSM) or to distinguish two states of a DFSM [33]. It is also known that the corresponding problems are EXPTIME-complete for finite state machines (FSMs) that need not be deterministic [4]. In addition, they are undecidable for distributed testing, in which the SUT has multiple interfaces and a separate tester is placed at each interface [25]. In this paper we investigate the problems of reaching a state, executing a transition and distinguishing two states for asynchronous testing through channels. We consider two types of channels: those that are first-in-first-out (FIFO) and non-FIFO channels. Previous work has assumed that channels are FIFO [29, 31, 14] but in some situations, such as communicating through the Internet, channels are non-FIFO. Since the work on asynchronous testing has considered testing from an IOTS [28, 29, 30, 14], and IOTSs are more general than FSMs, we use this formalism.

The problems of reaching states, executing transitions and distinguishing states are all highly relevant to test case generation. However, the use of a model such as an IOTS provides further opportunities for test automation: there is also the potential to check the observed behaviour against the model. This problem, of deciding whether an observed sequence of inputs and outputs is consistent with the model, is called the *Oracle Problem*. Automated solutions to the Oracle Problem can make testing significantly cheaper and allow many more test cases to be used. This has led to significant interest in the Oracle Problem [48, 10, 11, 12]. In this paper we investigate the Oracle Problem for asynchronous testing.

In this paper we use a general notion of a test case that allows it to be adaptive: we do not restrict attention to sequences. The main results in this paper are as follows. We prove that we can decide in polynomial time whether there is a test case that is guaranteed to reach a given state, execute a given transition, or distinguish two given states when the IOTS is observable. An IOTS M is observable if it has no state s and input or output a such that M can move from s to states s_1 and s_2, $s_1 \neq s_2$, under a and so this does not require M to be deterministic. We also prove that the problems are EXPTIME-hard if we do not require the IOTSs to be observable. In contrast, with non-FIFO channels it transpires that the problems of deciding whether there is a test case that reaches a state, executes a transition or distinguishes two states are EXPTIME-hard even for deterministic IOTSs. In addition, for non-FIFO channels the Oracle Problem is NP-complete, again even if we restrict attention to deterministic IOTSs. These results suggest that testing through asynchronous channels is significantly more difficult when the channels are non-FIFO than when they are FIFO.

This paper is structured as follows. We define IOTSs in Section 2 and in Section 3 we describe the types of test cases we consider. We investigate the problems of reaching states, executing transitions and distinguishing states in Sections 4, 5, and 6 respectively. We then explore the Oracle Problem in Section 7. Finally, conclusions are drawn in Section 8.

2. Input Output Transition Systems

An Input Output Transition System (IOTS) is defined by a set of states with transitions between them. Each transition has a label that is either an input or an output[1]. Since we are interested in defining algorithms and deciding complexity, we restrict attention to IOTSs that have finite sets of states, inputs and outputs. An Input Output Transition System M is thus defined by a tuple (S, s_0, I, O, h) in which S is a finite set of states; $s_0 \in S$ is the initial state; I is the finite input alphabet; O is the finite output alphabet; and h is the transition relation of type $S \times (I \cup O) \leftrightarrow S$. For $s \in S$ and $a \in I \cup O$ we let $h(s, a)$ denote the set of $s' \in S$ such that (s, a) and s' are related under h. If $s' \in h(s, a)$ for $a \in I \cup O$ then M can move from state s to state s' through action a and this defines a *transition* (s, a, s') with starting state s, ending state s', and label a. A transition with the same starting and ending states is said to be a *self-loop transition.*

We use the usual convention in which the name of an input is preceded by ? and the name of an output is preceded by !. An IOTS is *output divergent* if it has a state from which it is possible to take an infinite sequence of consecutive transitions whose labels are outputs. Output divergence is similar to a livelock and is usually undesirable. Similar to [28, 29, 14], we therefore assume that any IOTS considered is not output divergent. An IOTS is *input-enabled* if for every

[1]Sometimes internal actions, with label τ, are allowed. However, to simplify the exposition we will do not consider internal actions.

state and input there is at least one associated transition: for all $s \in S$ and $?i \in I$, $h(s, ?i) \neq \emptyset$. A state s of M is said to be *stable* if for all $!o \in O$ we have that $h(s, !o) = \emptyset$. Thus, a state s of M is stable if it is not possible for M to leave s without M receiving an input.

This paper only considers input-enabled IOTSs and we will use M to refer to such an IOTS. If the IOTS used is not input-enabled then the semantics of the language from which the IOTS was produced often allows the IOTS to be completed. For example, for each state s and input $?i$ such that $h(s, ?i) = \emptyset$ we might add a transition $(s, ?i, s)$ (the input of $?i$ has no effect), which corresponds to the semantics given in Harel Statecharts [20], or we might add a transition from s with input $?i$ to an error state. We say that an IOTS M is *observable* if h is a function [37] (for all $s \in S$ and $a \in I \cup O$, $|h(s, a)| \leq 1$) although the term output-deterministic has also been used [4]. We can convert an IOTS into an observable IOTS using the standard algorithm for converting a finite automaton into a deterministic finite automaton [40], although this can lead to an exponential increase in the number of states. We will say that M is *deterministic* if h is a function, and for all $s_1, s, s' \in S$ and outputs $!o' \neq !o$, we cannot have that $(s_1, !o, s)$ and $(s_1, !o', s')$ are transitions of M. A recent piece of work [14] assumes that quiescence can be observed, where the SUT is quiescent if it is in a state that it cannot leave without first receiving input (a stable state). It seems likely that the observation of quiescence, which is typically through a timeout, is feasible for some systems that communicate asynchronously but not others. We do not assume that quiescence can be observed but the observation of quiescence has very little effect on the results in this paper.

An IOTS M interacts with its environment through a sequence of steps where each step involves M receiving an input or producing an output, a step corresponding to a transition of M. M thus interacts with its environment through a sequence of consecutive transitions. Such a sequence $\rho = t_1 \ldots t_k$, $t_i = (s_i, a_i, s_{i+1})$, is a *walk* with *label* $a_1 \ldots a_k$, *starting state* s_1 and *ending state* s_{k+1}. If the starting state of ρ is s_0 then $a_1 \ldots a_k \in (I \cup O)^*$ is said to be a *trace* of M.

An IOTS M defines the regular language $L(M)$ of labels of walks with starting state s_0: the set of *traces* of M. Similarly, we let $L_M(s)$ denote the set of labels of walks of M with starting state s. Two states s and s' of M are said to be *equivalent* if they define the same languages: $L_M(s) = L_M(s')$. Similarly, two IOTSs M and N are *equivalent* if $L(M) = L(N)$.

3. Using strategies for testing

Most work on testing from deterministic models assumes that a preset input sequence is applied: for deterministic systems less is gained by using an adaptive process since as soon as the behaviour of the SUT diverges from that of M we know that there has been a failure. In the context of testing from a nondeterministic FSM there has been interest in adaptive testing (see, for example, [4, 22, 23, 24, 33, 44, 49]).

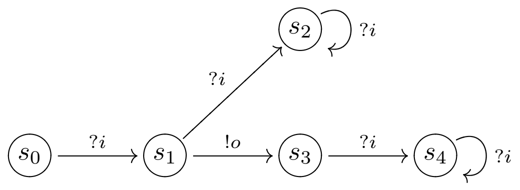

Figure 1: A deterministic IOTS with two possible responses to $?i?i$

In testing with asynchronous channels we may want testing to be adaptive, even if M is deterministic. This is because the tester cannot directly observe the inputs and outputs of the SUT but instead observes the sequence of inputs and outputs in a system composed of the SUT and channels. As a result, inputs arrive at the SUT later than they were sent by the tester and the observation of an output by the tester will be later than the generation of this output by the SUT. Thus, if the tester has observed a trace $a_1 \dots a_k$ then this may not be a trace of the SUT and, in addition, the SUT may have produced further outputs that have yet to be observed. This effectively introduces nondeterminism into testing. Let us suppose, for example, that we supply $?i?i$ when the IOTS shown in Figure 1 is in state s_0. There are two possibilities regarding the second $?i$: it is received before $!o$ is output by the SUT and so the IOTS ends in state s_2 or it is received after $!o$ is output by the SUT and so the IOTS ends in state s_4.

A strategy specifies what input the tester should supply and when. We will assume that the tester uses a strategy μ that is a partial function from $(I \cup O)^*$ to I and so it makes decisions, regarding the sending of input, on the basis of the observations it has made. The tester uses the strategy μ in the following way: if the current sequence of observations is σ and $\mu(\sigma)$ is defined and equals $?i \in I$ then the tester sends $?i$ to the SUT and otherwise it does nothing. The SUT receives the input of $?i$ at some later point and this triggers a transition. Since we are interested in the use of strategies in testing, we use the terms *strategy* and *test case* interchangeably.

Consider now the nondeterministic IOTS in Figure 2 and let us suppose that we wish to reach state s_3 in synchronous testing. After applying input $?i$ we either observe $!o_1$ and then apply $?i$ or we observe $!o_2$ and apply no further input. This defines a strategy that reaches (ends in) state s_3 but it is clear that there is no single sequence that is guaranteed to achieve this: the behaviour after the first input must depend on the resultant output.

We can define the set of possible sequences of interactions the tester can have with the composition of an SUT with channels given strategy μ.

Definition 1 *A sequence $\sigma \in (I \cup O)^*$ is* an evolution *of strategy μ if the following properties hold:*

1. *If $\sigma_1 ?i$ is a prefix of σ and $?i \in I$ then $\mu(\sigma_1) = ?i$. This says that the tester only sends an input when this is specified by the strategy.*

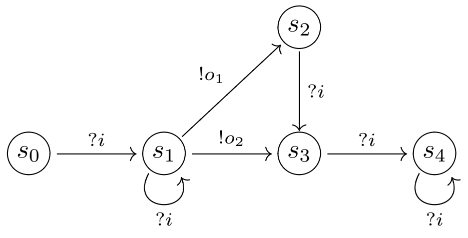

Figure 2: A nondeterministic IOTS

2. *If σ_1 is a proper prefix of σ and $\mu(\sigma_1) = ?i$ then $\sigma_1?i$ is a prefix of σ. This says that the tester sends an input whenever this is specified by the strategy.*

We let $Ev(\mu)$ denote the set of evolutions of μ.

While this defines the set of possible observable traces when using strategy μ, it does not necessarily correspond to the possible traces that M can perform when using μ since the use of asynchronous channels can mean that the observed trace σ was not the trace of M that occurred. We therefore need to reason about the relationship between traces that are observed and those that can be produced by the SUT.

First consider the case where communications are through FIFO channels. We know that if trace σ' of M occurred then the observed trace σ can only differ from σ' through σ being formed from σ' by the delaying of output. We therefore say that σ can result from σ', denoted $\sigma \preceq \sigma'$, if σ can be formed from σ' through a sequence of transformations of the form $!o?i \rightarrow ?i!o$ for $?i \in I$ and $!o \in O$.

Definition 2 *A sequence $\sigma' \in (I \cup O)^*$ is* an internal evolution *of strategy μ* with FIFO channels *if there exists $\sigma \in Ev(\mu)$ such that $\sigma \preceq \sigma'$. We let $Int(\mu)$ denote the set of internal evolutions of μ with FIFO channels. Where it is clear from the context that the channels are FIFO we simply say that σ' is* an internal evolution *of μ*

The set of internal evolutions of μ with FIFO channels are the traces that an SUT might produce when the tester is using strategy μ and the tester and SUT interact through FIFO channels. The idea is that any interaction σ between the system composed of the SUT and the channels must be an evolution of μ but the actual trace of the SUT could be any σ' with $\sigma \preceq \sigma'$.

Now consider the case where the channels are non-FIFO. We know that if trace σ' of M occurred then the observed trace σ can differ from σ' through σ' being formed from σ by the delaying of output in σ, by inputs arriving in a different order to which they were sent or by outputs arriving in a different order to which they were sent. We therefore say that observed sequence σ can result from the trace σ' of the SUT, denoted $\sigma \sqsubseteq \sigma'$, if σ can be formed from σ' through a sequence of transformations of the form: $!o?i \rightarrow ?i!o$ for $?i \in I$ and $!o \in O$; $!o_1!o_2 \rightarrow !o_2!o_1$ for $!o_1, !o_2 \in O$; $?i_1?i_2 \rightarrow ?i_2?i_1$ for $?i_1, ?i_2 \in I$.

Definition 3 *A sequence $\sigma' \in (I \cup O)^*$ is* an internal evolution *of strategy μ* with non-FIFO channels *if there exists $\sigma \in Ev(\mu)$ such that $\sigma \sqsubseteq \sigma'$. We let $Int'(\mu)$ denote the set of internal evolutions of μ with non-FIFO channels. Where it is clear from the context that the channels are non-FIFO we simply say that σ' is* an internal evolution *of μ*

In this paper we consider the problem of defining strategies for three goals: reaching a state of an IOTS, executing a transition of an IOTS, and distinguishing two states of an IOTS. The first two goals refer to the trace that the SUT performed, not that observed, and so we will have to reason about internal evolutions. Later we formally define what it means to reach a state of an IOTS (Definitions 4 and 6), to execute a transition of an IOTS (Definitions 7 and 8) or to distinguish two states of an IOTS (Definitions 10 and 12). As discussed earlier, all three problems are motivated by standard test techniques and criteria.

Strategies similar to those described above have been studied in the context of two player games. Alur et al. [4] expressed the problems of reaching and distinguishing states of non-deterministic FSMs in terms of two player games with incomplete information and we use results regarding such games to reason about the complexity of problems in asynchronous testing.

A two player $\exists\forall$ game G with incomplete information is a nondeterministic machine $(Q, X, Y, k, Q_{in}, Q^f)$ in which Q is a set of game positions, X is the set of inputs, Y is the set of outputs, k is a transition relation of type $Q \times X \leftrightarrow Q \times Y$, Q_{in} is a set of starting positions and Q^f is a set of winning position. The game starts in one of the positions in Q_{in} and in each move the $\exists$ player chooses an input to apply and the $\forall$ player then chooses a transition of G to follow: if the game position before the move is q and the $\exists$ player applies input $?i$ then the $\forall$ player chooses the output $!o$ and next game position q' from those that satisfy $(q', !o) \in k(q, ?i)$. The $\exists$ player observes the output but not the game position and uses a $\exists$ strategy μ of type $O^* \rightarrow I$ that specifies the next input to apply. The game ends if it reaches a position in Q^f and the $\exists$ player then wins. If the game does not end then the $\forall$ player wins.

The *outcome problem* for a $\exists\forall$ game G with incomplete information is to determine whether there is a *winning strategy* for the $\exists$ player: a strategy for the $\exists$ player that guarantees that they will win. It has been proved that this outcome problem is EXPTIME-complete [41]. Naturally, in reasoning about such results we restrict attention to games in which the sets Q, X, Y are finite. At times we will restrict the sets Q_{in} and/or Q_f to contain a single state. However, restricting Q_{in} to contain a single state does not change the complexity of the outcome problem since we can add a new unique start state and the initial moves take the game to states in Q_{in}. Similarly, we can represent a game G with more than one winning state by adding a new input $?i$, new winning state

s_w and the following moves in response to $?i$: if $?i$ is applied in a state in $Q \setminus Q_f$ then there is a fixed output $!o \in O$; if $?i$ is applied in a state in Q_f then it takes the game to s_w.

An alternative approach to strategies is to define test cases as trees, called transfer trees [49]. It has been shown that for observable non-deterministic finite state machines it is possible to find optimal transfer trees that reach or distinguish states when communications are synchronous [49]. We will use these results despite the fact that we consider IOTSs rather than finite state machines and we also use asynchronous communications.

4. Reaching states

Alur et al. [4] showed that the problem of deciding whether there is a strategy that reaches a given state of a nondeterministic FSM, when communications are synchronous, is EXPTIME-complete. They achieve this by showing that the outcome problem for a $\exists\forall$ game G can be converted into a problem of finding a strategy to reach a state of a nondeterministic FSM. In contrast, if the IOTS is deterministic, and communications are synchronous, then we can solve the problem through the use of a depth-first search and this takes linear time [43].

The situation is different in asynchronous testing since a trace observed by the tester does not have to be a trace of the SUT. This introduces nondeterminism even when the SUT is deterministic: an input may lead to several possible outputs as a result of the different possible delays of the input. Indeed, we can simulate nondeterminism by introducing a sequence of k outputs by the SUT and the sending of an input $?i$: the input will arrive at the SUT after $k' \leq k$ of the outputs and different values of k' can lead to different behaviours. Our problem thus appears to be similar to that of reaching a state of a nondeterministic FSM, a problem that is EXPTIME-hard. However, we will see that we can solve this problem in polynomial time when the IOTS is observable and we have FIFO channels while it is EXPTIME-hard when the channels are non-FIFO.

First we consider the problem of finding a strategy to reach a state s of IOTS M when communications are asynchronous through FIFO channels. By reaching a state we mean that at the end of the application of the test case (strategy) μ we must have that M is in state s. As a result, only stable states are reachable. Consider again the IOTS shown in Figure 1 in which the input $?i$ takes the IOTS to state s_1. We do not say that s_1 is reachable because it is not a stable state; from s_1 the IOTS can move to s_3 under output $!o$. As a result, state s_3 is reachable using input $?i$. Since s_3 is reachable, so is s_4: we apply input $?i$ and then supply $?i$ again after $!o$ is observed. In contrast, s_2 is not reachable since if we supply input sequence $?i?i$ we cannot guarantee that the second $?i$ arrives before $!o$ is produced.

There is an alternative notion of μ reaching state s of M: its application must lead to M entering state s but M need not be in state s at the end of the application of μ. However, if we are interested in this alternative notion of reaching s then we can rewrite M to form an IOTS M^s by removing all transitions with starting state s and adding self-loops with input in state s.

Then, μ reaches state s of M in the alternative notion of reachable if and only if μ reaches state s of M^s using the notion of reachable that we use. Thus, we lose nothing in requiring that M is in state s at the end of the application of μ: we can transfer the results to the alternative notion of reaching a state.

Before considering the problem of reaching states in detail we define what it means for a strategy μ to reach a stable state s of M. Essentially, we require that if σ is an internal evolution of μ, and so is consistent with some evolution of μ, and it is also a possible trace of M then all walks of M with label σ have ending state s.

Definition 4 *Strategy μ* reaches *the state s of M with FIFO channels if s is a stable state, there is an upper bound on the length of the sequences in $Int(\mu) \cap L(M)$ and for all $\sigma \in Int(\mu) \cap L(M)$ that label maximal walks of M that end in stable states, we have that every walk of M from state s_0 with label σ has ending state s.*

In this definition we only consider the labels in $Int(\mu) \cap L(M)$ of maximal walks (those that are not proper prefixes of other walks with labels in $Int(\mu) \cap L(M)$) since we are concerned with the state of M *after* μ has been applied and not the states of M met earlier. We require there to be an upper bound on the length of such traces since we want the application of μ to terminate.

We now investigate the problem of deciding whether such a strategy exists for a given state s of a deterministic IOTS M; later in this section we consider cases where M is nondeterministic. First we prove that it is sufficient to consider a special type of strategy when M is observable.

Proposition 1 *There is a strategy μ that reaches a state s of an observable IOTS M in asynchronous testing with FIFO channels if and only if there is such a strategy μ' that only applies inputs in stable states.*

Proof
Since M can be nondeterministic there may be several initial output sequences. Consider one possible first input $?i$, which μ sends after output sequence σ is observed. Further, let us suppose that M has a walk from its initial state to a stable state whose label is the output sequence σ' such that σ is a proper prefix of σ'. Since communications are asynchronous and so an input is delayed by the channel, $?i$ might arrive at the SUT after any σ'' has been produced by the SUT such that σ is a prefix of σ'' and σ'' is a prefix of σ'. In addition, in each possible case μ must lead to state s eventually being reached. In particular, μ must define a strategy for the case where $?i$ arrives after σ' and so when $?i$ arrives in a stable state. In addition, since M is observable we know when a stable state has been reached: we cannot have the situation in which there are two walks from a state s' with the same output sequence as label and one reaches a stable state but the other does not. We can therefore produce a strategy μ_1 from μ by requiring that if an output sequence with prefix σ occurs then $?i$ is applied in a stable state. We can now repeat this process to obtain a strategy in which inputs are only applied in stable states.

The converse direction follows immediately and so the result holds. □

This approach does not work if M is not observable since the tester need not know when the SUT is in a stable state. For example, from a state s there may be two transitions with label $!o$, one of which takes M to a stable state while the other takes M to a state that is not stable.

We can assume that the state s is a stable state: otherwise it is not reachable. We now define an IOTS that corresponds to M only receiving input in stable states.

Definition 5 *Given an IOTS $M = (S, s_0, I, O, h)$ we define the IOTS $S(M) = (S, s_0, I, O, h')$ in which the transitions of $S(M)$ are defined by the following rules.*

- *For each $s \in S$ and $!o \in O$, $h'(s, !o) = h(s, !o)$.*
- *For each stable state $s \in S$ and $?i \in I$, $h'(s, ?i) = h(s, ?i)$.*
- *For each state $s \in S$ that is not a stable state and $?i \in I$, $h'(s, ?i) = \emptyset$.*

By Proposition 1 it is sufficient to consider $S(M)$ when deciding reachability for deterministic IOTSs.

Theorem 1 *The problem of deciding whether there exists a strategy that reaches a state s of a deterministic IOTS M in asynchronous testing with FIFO channels can be solved in linear time.*

Proof
Assume that s is a stable state of M since otherwise we immediately know that it is not reachable. By Proposition 1 it is sufficient to consider strategies in which inputs are applied in stable states. Since M is deterministic, $S(M)$ has a unique stable state reached without input and this is the state in which the first input is applied. Since M is deterministic, $S(M)$ is also deterministic. In addition, stable state s is reachable if and only if it is reachable in $S(M)$ from the state s_0. We can consider $S(M)$ to be a directed graph $\mathcal{G}$: the states of $S(M)$ are represented by the vertices of $\mathcal{G}$ and each transition of $S(M)$ is represented by an edge. Then, s is reachable if and only if there is a walk in $\mathcal{G}$ from the vertex representing s_0 to the vertex representing s and we can use a depth-first search to decide this in linear time [43]. The result therefore holds. □

Now consider the case where M is observable but need not be deterministic. Clearly $S(M)$ need not be deterministic and there is a set S^0 of states in which the first input can be applied: the stable states of M that are reached from s_0 without applying input. However, $S(M)$ can be seen as an observable nondeterministic finite state machine and so we can decide in polynomial time whether there is a transfer tree that reaches a given state s [49]. Since we require the application of strategies to lead to a bounded set of possible sequences, strategies correspond exactly to transfer trees and so the following holds.

Theorem 2 *The problem of deciding whether there exists a strategy that reaches a state s of an observable IOTS M in asynchronous testing with FIFO channels can be solved in polynomial time.*

Finally, we consider the general case in which M need not be observable.

Theorem 3 *The problem of deciding whether there exists a strategy that reaches a state s of an IOTS M in asynchronous testing with FIFO channels is EXPTIME-hard.*

Proof
Recall that a two player $\exists\forall$ game G with incomplete information is a nondeterministic machine $(Q, X, Y, k, Q_{in}, Q^f)$ in which Q is a set of game positions, X is the set of inputs, Y is the set of outputs, k is a transition relation of type $Q \times X \leftrightarrow Q \times Y$, Q_{in} is a set of starting positions and Q^f is a set of winning position. In addition, as explained in Section 3, we can assume that Q_{in} contains only one state, which we call q_{in} and also that the set Q^f of winning states also only contains one state, which we call q^f. Such a game G defines an IOTS M in which we complete (Q, q_{in}, X, Y, k) by, for each state s' reached by a transition labelled by an input, adding transitions from s' labelled with inputs to an error state s_e from which we cannot reach q^f. A strategy for the $\exists$ player is a winning strategy if it is guaranteed to take G to state q^f. Now observe that a strategy that is guaranteed to take M to q^f must have input and output alternating, since otherwise the error state might be reached, and so a strategy will involve applying an input, waiting for the output, and then applying the next input. This corresponds exactly to a strategy for G and so a strategy μ takes M to q^f if and only if μ is a winning strategy for G. The result now follows from the fact that we can construct M from G in polynomial time and the problem of deciding whether there is a winning strategy for a two player $\exists\forall$ game with incomplete information is EXPTIME-hard [41]. □

Now consider non-FIFO channels.

Definition 6 *Strategy μ* reaches *the state s of M with non-FIFO channels if s is a stable state, there is an upper bound on the length of the sequences in $Int'(\mu) \cap L(M)$ and for all $\sigma \in Int'(\mu) \cap L(M)$ that label maximal walks of M that end in stable states, we have that every walk of M from state s_0 with label σ has ending state s.*

The use of non-FIFO channels introduces an important difference: if from a stable state we send a sequence σ_1 of inputs before waiting for output (i.e. moving from stable states to other stable states) then these inputs might be received in any order. This allows us to simulate non-determinism in a $\exists\forall$ game in the following way.

Theorem 4 *For asynchronous testing with non-FIFO channels, the problem of deciding whether there is a strategy that reaches a state s of IOTS M is*

EXPTIME-hard. In addition, this result holds even if we restrict M to being deterministic.

Proof
We will prove this by showing that an instance of an $\exists\forall$ game with incomplete information can be converted into the problem of reaching a state of an IOTS through non-FIFO channels. We therefore assume that we have $\exists\forall$ game G with incomplete information that has winning state s.

Below we form an IOTS $\mathcal{M}(G)$, with a new input $?i_0$, where a strategy that reaches state s defines a winning strategy for G. We also have a new state s_e that will represent an 'error state': from s_e ($s_e \neq s$) all transitions are self-loops labelled with input and so it is not possible to get from s_e to s. Let d denote an upper bound on the degree of branching in G: for every state s of G and input $?i$, the $\forall$ player has no more than d possible responses to the $\exists$ player supplying $?i$ in state s.

Consider a state s_i of G and input $?i$ that could be supplied by the $\exists$ player. and assume that the $\forall$ player has $d' \leq d$ possible responses. Then we introduce a walk $(s_i^1, ?i_0, s_i^2), (s_i^2, ?i_0, s_i^3) \ldots (s_i^{d-1}, ?i_0, s_i^d)$ in which $s_i^1 = s_i$. We order the possible responses of the $\forall$ player to $?i$ in state s_i and let us suppose that the jth possible response involves output $!o_j$ and moving to state u_i^j. We will use the arrival of $?i$ in state s_i^j to represent the jth possible move for the $\forall$ player and include a transition $(s_i^j, ?i, t_i^j)$. If $d' < d$ then for all $d' \leq j < d$ we let the jth choice be arbitrarily set to one of the allowed responses of the $\forall$ player. From t_i^j we include a walk whose label has $d - j$ instances of $?i_0$ followed by output $!o_j$ and $\mathcal{M}(G)$ moving to state u_i^j. Thus, each possible response to $?i$ is triggered from at least one point at which $?i$ might arrive in an input sequence consisting of $?i$ and $d-1$ instances of $?i_0$. If any input other than $?i_0$ is received in one of these states, on the walk from t_i^j to u_i^j, then $\mathcal{M}(G)$ moves to state s_e. Note that when considering different inputs in s_i we use the same intermediate states of the form s_i^j. This ensures that $\mathcal{M}(G)$ is deterministic. We now complete $\mathcal{M}(G)$ to make it input-enabled by making the input of $?i_0$ in a state s_i^d of $\mathcal{M}(G)$ lead to s_e.

Now consider a strategy μ that moves $\mathcal{M}(G)$ to state s. Then μ must operate in the following way: apply an input $?i$ and also apply $?i_0$ a total of $d-1$ times. The next input, if any, is determined by the output produced. Each possible response of the $\forall$ player is simulated by a possible number of $?i_0$ that arrive before $?i$ and so a strategy for reaching s defines a winning strategy for G. Thus, a strategy reaches state s of $\mathcal{M}(G)$ if and only if it defines a winning strategy of G. Thus, the result follows from the outcome problem for $\exists\forall$ games with incomplete information being EXPTIME-complete [41]. $\square$

5. Executing transitions

One of the standard test criteria used when testing from a state machine is transition coverage: we wish to apply a set of test cases that is guaranteed to lead to every transition being executed. This criterion has been explored in the context of asynchronous testing with FIFO channels [29] but algorithms were not given for generating test cases that satisfy this criterion[2] and so complexity issues were not considered. In order to execute a transition t of M we need to take M to the starting state of t and then apply the input of t. As a result, it might initially seem that this is identical to reachability but there is a difference: when executing a transition we do not require that the strategy terminates once the objective has been achieved.

In this section we prove that we can represent the problem of reaching a state in terms of executing a transition and that the converse also holds. Before doing this we define what it means for a strategy to execute a transition when there are FIFO channels and also when there are non-FIFO channels.

Definition 7 *Strategy μ* executes transition *t of M with FIFO channels if there is an upper bound on the length of sequences in $Int(\mu) \cap L(M)$ and for all $\sigma \in Int(\mu) \cap L(M)$ that is not a proper prefix of another sequence in $Int(\mu) \cap L(M)$ we have that every walk of M from state s_0 with label σ contains t.*

Definition 8 *Strategy μ* executes transition *t of M with non-FIFO channels if there is an upper bound on the length of sequences in $Int'(\mu) \cap L(M)$ and for all $\sigma \in Int'(\mu) \cap L(M)$ that is not a proper prefix of another sequence in $Int'(\mu) \cap L(M)$ we have that every walk of M from state s_0 with label σ contains t.*

First we show how we can express the problem of executing a transitions t in terms of reachability. For transition $t = (s, x, s')$ of M we form a new IOTS M_t by adding a new state s_t, change t to $t' = (s, x, s_t)$ and in s_t having a self-loop $(s_t, ?i, s_t)$ for every input $?i$. The following properties are clear from the definition of M_t, with the only differences between M and M_t occurring *after* the execution of t in M and t' in M_t.

Proposition 2 *Let us suppose that $t = (s, x, s')$ is a transition of IOTS M and t' is the transition (s, x, s_t) of M_t. Then the following hold for both FIFO and non-FIFO channels:*

1. *Given a strategy μ, μ executes t in M if and only if μ executes t' in M_t.*
2. *Given a strategy μ, μ executes t' in M_t if and only if μ reaches state s_t in M_t.*

Then any strategy that reaches s_t in M_t executes t in M and any strategy that executes t in M reaches s_t in M_t. This leads to the following results.

[2]Instead, the authors considered the problem of deciding whether a given input sequence, which executes a transition t when using synchronous testing, ensured that t is executed when using FIFO channels.

Theorem 5 *The problem of deciding whether there exists a strategy that executes a transition t of a deterministic IOTS M in asynchronous testing with FIFO channels can be solved in linear time.*

Proof
We can construct deterministic M_t in constant time. From Proposition 2 we know that a strategy leads to t being executed in M if and only if it reaches the state s_t in M_t. The result thus follows from Theorem 1. $\square$

Theorem 6 *The problem of deciding whether there exists a strategy that executes a transition t of an observable IOTS M in asynchronous testing with FIFO channels can be solved in polynomial time.*

Proof
We can construct observable M_t in constant time and we know that a strategy leads to t being executed in M if and only if it leads to the state s_t in M_t. The result thus follows from Theorem 2. $\square$

We now show how we can express the problem of reaching a stable state s of M in terms of covering a transition. In order to achieve this we produce an IOTS M_s from M by introducing a new input $?i_s$ such that $?i_s$ takes M_s from s to a new state s_p and takes every other state to an error state s_e. The only transitions from s_e and s_p are self-loops labelled with input. We do this for the special case in which there are no output sequences of length more than 1 and all transitions that reach s produce output. This condition is satisfied by the IOTS used in the proofs of Theorems 3 and 4 and ensures that the tester knows when the strategy is complete; there are no more outputs to be produced.

Proposition 3 *Let us suppose that s is a state of IOTS M, every transition of M that ends in s has a label that is an output, there are no pairs of consecutive transitions of M that are labelled with output, and the channels are FIFO or non-FIFO. There exists a strategy that executes transition $(s, ?i_s, s_p)$ of M_s if and only if there is a strategy that reaches state s of M.*

Proof
First note that if a strategy μ reaches the state s of M then μ followed by $?i_s$ executes the transition $(s, ?i_s, s_p)$ of M_s. Further, if μ executes the transition $(s, ?i_s, s_p)$ of M_s then it must apply $?i_s$ in state s and not before, since otherwise it might enter the error state. Thus, μ, with all instances of $?i_s$ and later input removed, reaches the state s of M. The result therefore holds. $\square$

Theorem 7 *The problem of deciding whether there exists a strategy that executes a transition t of an IOTS M in asynchronous testing with FIFO channels is EXPTIME-hard.*

Proof
This follows from Proposition 3 and Theorem 3. $\square$

Now consider the case where channels are non-FIFO.

Theorem 8 *The problem of deciding whether there exists a strategy that executes a transition t of an IOTS M in asynchronous testing with non-FIFO channels is EXPTIME-hard. In addition, this result still holds if we restrict M to being deterministic.*

Proof
First consider a game G and with $M = \mathcal{M}(G)$ as constructed in the proof of Theorem 4. Then we have that both M and M_s are deterministic. The result thus follows from Proposition 3 and Theorem 4. $\square$

6. Distinguishing states

In this section we consider the problem of distinguishing two states of an IOTS. In the previous sections we were concerned with the trace of M and what we can deduce about this from an observed trace σ. However, we can only distinguish two states through external observations and thus not directly from traces of M. We therefore first define the set of traces that can be observed when interacting with M through channels: the traces that can be formed from traces of M through delaying outputs.

Definition 9 *A sequence $\sigma \in (I \cup O)^*$ is* an external trace *of IOTS M in state s with FIFO channels if there exists $\sigma' \in L_M(s)$ such that $\sigma \preceq \sigma'$. We let $\mathcal{E}(M,s)$ denote the set of external traces of M in state s with FIFO channels. Further, given IOTS M, state s of M, and strategy μ, we let $\mathcal{T}r(M,\mu,s)$ be the maximal elements of $\mathcal{E}(M,s) \cap Ev(\mu)$: those that are not proper prefixes of sequences in $\mathcal{E}(M,s) \cap Ev(\mu)$.*

Thus, $\mathcal{T}r(M,\mu,s)$ is the set of (complete) traces that can be observed when testing M using μ when the testing starts in state s.

We can now define what it means to distinguish between two states when there are FIFO channels.

Definition 10 *A strategy μ* distinguishes *states s and s' of IOTS M with FIFO channels if there is an upper bound on the lengths of the traces in $\mathcal{T}r(M,\mu,s)$ and $\mathcal{T}r(M,\mu,s')$ and $\mathcal{T}r(M,\mu,s) \cap \mathcal{T}r(M,\mu,s') = \emptyset$.*

This says that when using strategy μ, if σ is a possible observation when applying μ from state s then it is not a possible observation when applying μ from state s' and that the converse also applies. We can now extend this to non-FIFO channels.

Definition 11 *A sequence $\sigma \in (I \cup O)^*$ is* an external trace *of IOTS M in state s with non-FIFO channels if there exists $\sigma' \in L_M(s)$ such that $\sigma \sqsubseteq \sigma'$. We let $\mathcal{E}'(M,s)$ denote the set of external traces of M in state s with non-FIFO channels. Further, given IOTS M, state s of M, and strategy μ, we let $\mathcal{T}r'(M,\mu,s)$ be the maximal elements of $\mathcal{E}'(M,s) \cap Ev(\mu)$: those that are not proper prefixes of sequences in $\mathcal{E}'(M,s) \cap Ev(\mu)$.*

We can now define what it means to distinguish between two states when the channels are non-FIFO.

Definition 12 *A strategy μ* distinguishes *states s and s' of IOTS M with non-FIFO channels if there is an upper bound on the lengths of the traces in $\mathcal{T}r'(M, \mu, s)$ and $\mathcal{T}r'(M, \mu, s')$ and $\mathcal{T}r'(M, \mu, s) \cap \mathcal{T}r'(M, \mu, s') = \emptyset$.*

As before, if M is observable and we are using FIFO channels then it is sufficient to consider only strategies that apply input in stable states.

Proposition 4 *There is a strategy μ that distinguishing states s_1 and s_2 of an observable IOTS M in asynchronous testing with FIFO channels if and only if there is such a strategy μ' that only applies inputs in stable states when applied to s_1 or s_2.*

Proof
If s_1 and s_2 are not stable states then in each case there will be an output sequence before a stable state is reached and we can consider all such output sequences. If different output sequences are produced from s_1 and s_2, when reaching stable states, then there is no need to apply any further input. Thus, we only need to consider identical initial output sequences from s_1 and s_2 and since M is observable we know when a stable state has been reached. Now assume that for s_1 and s_2 we have that the output sequence σ takes M to stable states. Consider one possible first input $?i$ sent when μ is used to test M such that $?i$ is sent after an output sequence σ' that is a prefix of σ. As with the proof of Proposition 1, we cannot know that $?i$ is not applied in a stable state. Thus, μ must define a strategy for the case where $?i$ arrives after σ and so when $?i$ arrives in a stable state. We can therefore produce a strategy μ_1 from μ by requiring that $?i$ is applied in a stable state. We can now repeat this process to obtain a strategy in which inputs are only applied in stable states.

The converse direction follows immediately and so the result holds. $\square$

Given state s of M let $M(s)$ denote the IOTS formed by making s the initial state of M. Given states s_1 and s_2 we can form the IOTS $S(M(s_1))$ and $S(M(s_2))$ that represent the behaviours of $M(s_1)$ and $M(s_2)$ respectively when input is only applied in stable states.

The following is clear.

Proposition 5 *With FIFO channels, there is a strategy that distinguishes states s_1 and s_2 of observable IOTS M if and only if there is a strategy that distinguishes $S(M(s_1))$ and $S(M(s_2))$.*

Definition 13 *Given two observable IOTSs $M_1 = (S_1, s_0^1, I, O, h_1)$ and $M_2 = (S_1, s_0^2, I, O, h_2)$ the product $P(M_1, M_2)$ is the tuple $((S_1 \times S_2) \cup \{s_f\}, (s_0^1, s_0^2), I, O, h)$ in which s_f is not a state of M_1 or M_2 and the relation h is defined by the following.*

1. *Given $(s_1, s_2) \in S_1 \times S_2$ and $a \in I \cup O$, if $h_1(s_1, a) = \{s_1'\}$ and $h_2(s_2, a) = \{s_2'\}$ then $h((s_1, s_2), a) = \{(s_1', s_2')\}$.*

2. *Given $(s_1, s_2) \in S_1 \times S_2$ and $!o \in O$, if $h_1(s_1, !o) \neq \emptyset$ and $h_2(s_2, !o) = \emptyset$ or $h_1(s_1, !o) = \emptyset$ and $h_2(s_2, !o) \neq \emptyset$ then $h((s_1, s_2), !o) = \{s_f\}$.*
3. *The only transitions with starting state s_f are self-loops with input.*

The following is clear from the definition of the product $P(M_1, M_2)$ of IOTSs M_1 and M_2.

Proposition 6 *A strategy μ distinguishes states s_1 and s_2 of observable IOTS M in asynchronous testing with FIFO channels if and only if it reaches the state s_f of $P(S(M(s_1)), S(M(s_2)))$.*

Theorem 9 *Given two states s_1 and s_2 of an observable IOTS M, in asynchronous testing with FIFO channels we can decide in polynomial time whether there is a strategy that distinguishes s_1 and s_2.*

Proof
By Proposition 6 we know that it is sufficient to determine whether there is a strategy that reaches the state s_f of $P(S(M(s_1)), S(M(s_2)))$. In addition, it is clear that $P(S(M(s_1)), S(M(s_2)))$ is observable. If M has n states then $P(S(M(s_1)), S(M(s_2)))$ has $O(n^2)$ states and so the result follows from Theorem 2. □

Theorem 10 *Given two states s_1 and s_2 of an IOTS M, in asynchronous testing with FIFO channels the problem of deciding whether there is a strategy that distinguishes s_1 and s_2 is EXPTIME-hard.*

Proof
We will prove that if we can solve this then we can also solve the reachability problem. We therefore assume that we have an instance of the reachability problem in asynchronous testing with FIFO channels based on IOTS M and state s_k. We can assume that s_k is a stable state since otherwise we know that it is not reachable. We also assume that every transition of M that ends in s_k has a label that is an output and there are no pairs of consecutive transitions of M that are labelled with output. We can obtain a new IOTS M' from M in the following way:

1. For each state s_i of M we have two copies, s'_i and s''_i, the initial state being s'_0.
2. For $a \in I \cup O$ there is a transition (s''_i, a, s''_j) if and only if (s_i, a, s_j) is a transition of M.
3. For every transition (s_i, a, s_j) of M we have the transition (s'_i, a, s'_j).
4. We add a special input $?i_s$ such that from state s'_k the IOTS M' moves to a special state s_w and outputs a unique value $!o_w \notin O$, from every state of the form s'_i with $i \neq k$ there is a transition $(s'_i, ?i_s, s''_i)$ and from every state of the form s''_i there is a transition $(s''_i, ?i_s, s''_i)$.

This IOTS starts in state s'_0 and behaves like M until input $?i_s$ is received. If the machine is in state s'_i when $?i_s$ is first received then either $i = k$ and the special output $!o_w$ is produced and otherwise no output is produced and it moves to state s''_i. From there it behaves like M. Thus, a strategy μ distinguishes states s'_0 and s''_0 of this IOTS if and only if μ, with $?i_s$ and all input after $?i_s$ removed, reaches state s_k. Thus, by Theorem 3, the problem of deciding whether there is a strategy to distinguish states is EXPTIME-hard and so the result holds. □

When considering non-FIFO channels we get the following whose proof is equivalent to that of Theorem 10 except that it uses Theorem 4 rather than Theorem 3.

Theorem 11 *Given two states s_1 and s_2 of an IOTS M, in asynchronous testing with non-FIFO channels the problem of deciding whether there is a strategy that distinguishes s_1 and s_2 is EXPTIME-hard. In addition, this result holds even if we restrict M to being deterministic.*

7. The Oracle Problem

In testing, the problem of deciding whether an observation is consistent with the specification is called the Oracle Problem. We will assume that we have completed a test run and therefore that the SUT has reached a stable state and all output that has been produced has been observed. We are therefore interested in the walks of M that reach stable states and will let $L^\delta(M)$ denote the corresponding language: the set of traces in $L(M)$ that label walks from s_0 to stable states. It has been shown that given a sequence σ of observations, made when testing through FIFO channels, it is possible to define a finite automaton with $O(|\sigma|^2)$ states that accepts all sequences that the SUT might have produced: those in which the delay of output can lead to the observation of σ [14]. Further, this can be achieved in $O(|\sigma|^2)$ time. This leads to the following result.

Theorem 12 *Given $\sigma \in (I \cup O)^*$ with length k and IOTS M with p transitions, it is possible to decide whether there exists $\sigma' \in L^\delta(M)$ such that $\sigma \preceq \sigma'$ in $O(k^2p)$ time.*

Proof
We can produce a finite automaton N that accepts all σ' with $\sigma \preceq \sigma'$ in $O(k^2)$ time [14]. It is therefore sufficient to decide whether $L(N) \cap L^\delta(M)$ is empty and this can be decided in $O(k^2p)$ time by forming a finite automaton that accepts the language $L(N) \cap L^\delta(M)$. The result therefore holds. □

We now investigate the Oracle Problem for asynchronous testing with non-FIFO channels: the problem of deciding whether, for an observed sequence σ and IOTS M, there exists a sequence σ' that is consistent with σ ($\sigma \sqsubseteq \sigma'$) and that is in $L^\delta(M)$. We prove that this problem is NP-hard by showing that we can reduce the following to this problem.

Definition 14 *Given boolean variables $z_1, \ldots, z_r$ let $C_1, \ldots, C_k$ denote sets of three literals, where each literal is either a variable z_i or its negation. The three-in-one SAT problem is to decide whether there exists an assignment to the boolean variables such that each C_i contains exactly one true literal.*

The three-in-one SAT problem is known to be NP-hard [42].

Theorem 13 *Let us suppose that σ is a trace using the same input and output alphabets as the IOTS M and we are using asynchronous testing with non-FIFO channels. Then the problem of deciding whether there exists $\sigma' \in L(M)$ that can take M to a stable state such that $\sigma \sqsubseteq \sigma'$ is NP-complete. In addition, this result holds even if we restrict M to being deterministic.*

Proof

First we show that the problem is in NP by considering the following procedure. We first guess an order σ' of the elements of $E(\sigma)$, $\sigma = a_1, \ldots, a_k$, and check that this order is consistent with the partial order on the elements of σ: if $a_i \in O$ and $a_j \in I$ for $i < j$ then a_i must appear before a_j in σ'. If σ' passes this check then we determine whether $\sigma' \in L^{\delta}(M)$. Then, there exists $\sigma' \in L(M)$ that can take M to a stable state such that $\sigma \sqsubseteq \sigma'$ if and only if some guess σ' passes the two checks and since these checks can be performed in polynomial time we have that the Oracle Problem is in NP.

We now prove that the problem is NP-hard and assume that we have an instance of the three-in-one SAT problem with variables $z_1, \ldots, z_r$ and clauses $C_1, \ldots, C_k$ and show how this can be reduced to an instance of the Oracle Problem for asynchronous testing with non-FIFO channels.

We will define a deterministic IOTS M with two 'core' states, s_0, s_1, and 'error' states s_e^1 and s_e^2. The inputs of M are $?i_0, ?i_1, \ldots, ?i_r$ and the outputs of M are $!o_1, \ldots, !o_k, !o_e$.

If input $?i_j$ with $1 \leq j \leq r$ is received in state s_0 then the IOTS simulates the case where variable z_j is true by producing output $!o_p$ for each clause C_p that contains the literal z_j. This is achieved by a cycle that starts and ends at s_0 and starts with input $?i_j$.

Similarly, if input $?i_j$ with $1 \leq j \leq r$ is received in state s_1 then the IOTS simulates the case where variable z_j is false by producing output $!o_p$ for each clause C_p that contains the literal $\neg z_j$.

Input $?i_0$ moves M from state s_0 to state s_1 and in states s_1, s_e^1 and s_e^2 it leads to no change in state. The first $?i_0$ received, if received in state s_0, thus moves M from the state where inputs represent variables that are true to the state where inputs represent variables that are false.

If an input is received in a state other than s_0, s_1, s_e^1, or s_e^2 then it takes M to error state s_e^1. From state s_e^1 there is a transition with label $!o_e$ to s_e^2. The only other transitions from s_e^1 and s_e^2 are self-loops labelled with input.

Now consider $\sigma = ?i_0?i_1 \ldots ?i_r!o_1!o_2 \ldots !o_k$ and IOTS M. Since σ does not contain output $!o_e$, if there is $\sigma' \in L(M)$ that can take M to a stable state such that $\sigma \sqsubseteq \sigma'$ then we must have that the corresponding walk of M does not include state s_e^1. Thus, each input must be applied in either state s_0 or state s_1. In addition, the inputs might have arrived at M in any order. In particular, we cannot know whether an input $?i_j$, $1 \leq i \leq r$, was received before or after $?i_0$. Thus, an input $?i_j$, $1 \leq i \leq r$, might either have led to output $!o_p$ for each clause C_p that contains the literal z_j or might have led to output $!o_p$ for each clause C_p that contains the literal $\neg z_j$. In addition, the outputs are all observed after the inputs and so we cannot know which outputs were produced in response to particular inputs. The outputs $!o_1!o_2 \ldots !o_k$ also represent the case where each clause C_i contains exactly one literal that is true, for the values of the boolean variables represented by the transitions triggered by the $?i_j$. Thus, there exists $\sigma' \in L(M)$ that can take M to a stable state such that $\sigma \sqsubseteq \sigma'$ if and only if there is an assignment of truth values to variables $z_1, \ldots, z_r$ such that each clause C_p contains exactly one literal that is true. We have therefore reduced the three-in-one SAT problem to that of deciding whether there exists $\sigma' \in L(M)$ that can take M to a stable state such that $\sigma \sqsubseteq \sigma'$. The result now follows from observing that the three-in-one SAT problem is NP-hard and that both M and σ can be produced in polynomial time. $\square$

Now consider the alternative case where it is possible that not all outputs have been observed yet. We can adapt the proof that the problem is NP-hard as follows. Instead of considering an instance of the three-in-one SAT problem we consider an instance of the SAT problem with variables $z_1, \ldots, z_r$ and clauses $C_1, \ldots, C_k$. Here each clause is a set of literals and there is a solution to the SAT problem if and only if there is an assignment of values to the boolean variables $z_1, \ldots, z_r$ such that each clause contains at least one literal that is true. All other parts of the proof (that the problem is NP-hard) remain the same since for $\sigma = ?i_0?i_1 \ldots ?i_r!o_1!o_2 \ldots !o_k$ we have that there is some $\sigma' \in L(M)$ such that $\sigma \sqsubseteq \sigma'$ if and only if there is an assignment to the boolean variables such that each clause C_i contains at least one true literal (and so the sequence of outputs produced contains at least one instance of $!o_i$).

8. Conclusions

In model based testing (MBT) we base testing on a model M of the system under test or some aspect of the SUT. Typically, M is expressed as a state-based model such as a finite state machine or an input output transition system (IOTS) and communications are synchronous. Thus, many MBT approaches assume that the communications between the tester and the SUT are synchronous but there are important classes of system where this is unlikely to be the case. While we can compose M with models of the communications channels and then apply standard MBT approaches, this can lead to a significant increase in the state space.

In this paper we considered the problem of testing from an IOTS M where the SUT interacts with its environment through asynchronous channels. We considered the problem of producing strategies (test cases) that are *guaranteed* to reach a state, execute a transition or distinguish two states. We first considered the use of FIFO channels. We showed that the above problems can be solved in polynomial time for an observable model M. However, the general problems are EXPTIME-hard. While M can be used to drive test case generation, it can also be used to check whether an observed sequence of observations contains a failure (the Oracle Problem). When channels are FIFO the Oracle Problem can be solved in polynomial time.

We also investigated the case where the channels are non-FIFO. It transpired that all of the test case generation problems considered are EXPTIME-hard. In addition, the Oracle Problem is NP-complete. These results hold even if we restrict attention to the case where M is deterministic. It is clear that the test case generation problems are semi-decidable, however, we leave as an open question whether they are in EXPTIME.

Many systems interact with their environment through non-FIFO channels, an important class being any system that communicates via the Internet. The results in this paper suggest that non-FIFO channels introduce significant practical issues into testing. It may thus be best to test such systems without using the channels; by directly supplying input to the SUT and receiving output directly from the SUT. However, this is likely to be difficult for distributed systems and for such systems we may need to devise design for test guidelines that lead to systems that are easier to test.

There are several other possible lines of future work. For test case generation, we might also consider strategies that are *likely* to achieve an objective. In addition, while a number of the problems were NP-complete or EXPTIME-hard, it would be interesting to investigate reasonable conditions under which there are polynomial time solutions. Finally, it would be interesting to extend the results to certain types of model that have been proposed for distributed systems. Two types of model of particular note are those that use partial orders, rather than inputs and outputs, to label transitions [19, 46] and models in which the response to an input $?i$ in state s not being specified in model M means that if M receives $?i$ when in state s then $?i$ is retained in the input queue [29].

References

[1] H. AboElFotoh, O. Abou-Rabia, and H. Ural. A test generation algorithm for protocols modeled as non-deterministic FSMs. *The Software Engineering Journal*, 8(4):184–188, 1993.

[2] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. In *Protocol Specification, Testing, and Verification VIII*, pages 75–86, Atlantic City, 1988. Elsevier (North-Holland).

[3] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours. *IEEE Transactions on Communications*, 39(11):1604–1615, 1991.

[4] R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *27th ACM Symposium on Theory of Computing*, pages 363–372, 1995.

[5] M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Towards a tool environment for model-based testing with AsmL. In *Formal Approaches to Testing*, volume 2931 of *Lecture Notes in Computer Science*, pages 252–266, Montreal, Canada, 2003. Springer-Verlag.

[6] Puneet Bhateja, Paul Gastin, and Madhavan Mukund. A fresh look at testing for asynchronous communication. In *Automated Technology for Verification and Analysis, 4th International Symposium (ATVA 2006)*, volume 4218 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2006.

[7] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Asynchronous observations of processes. In *First International Conference on the Foundations of Software Science and Computation Structure (FoSSaCS 1998)*, volume 1378 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 1998.

[8] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. A theory of "may" testing for asynchronous languages. In *Second International Conference on the Foundations of Software Science and Computation Structure (FoSSaCS 1999)*, volume 1578 of *Lecture Notes in Computer Science*, pages 165–179. Springer, 1999.

[9] Ilaria Castellani and Matthew Hennessy. Testing theories for asynchronous languages. In *18th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1998)*, volume 1530 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 1998.

[10] T. Y. Chen, T. H. Tse, and Z. Zhou. Fault–based testing in the absence of an oracle. In *IEEE Annual International Computer Software and Applications Conference (COMPSAC 2001)*, pages 172–178. IEEE Computer Society Press, 2002.

[11] T. Y. Chen, T. H. Tse, and Z. Zhou. Fault-based testing in the absence of an oracle. *Information and Software Technology*, 45(1):1–9, 2003.

[12] Tsong Yueh Chen, Joshua W. K. Ho, Huai Liu, and Xiaoyuan Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinformatics*, 10, 2009.

[13] T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.

[14] Adenilso da Silva Simão and Alexandre Petrenko. Generating asynchronous test cases from test purposes. *Information and Software Technology*, 53:1252–1262, 2011.

[15] E. Farchi, A. Hartman, and S. Pinter. Using a model-based test generator to test for standard conformance. *IBM systems journal*, 41(1):89–110, 2002.

[16] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. In *Proceedings of the ACM SIGSOFT Symposium on Software Testing and Analysis*, pages 112–122, 2002.

[17] Wolfgang Grieskamp. Multi-paradigmatic model-based testing. In *Formal Approaches to Software Testing and Runtime Verification (FATES/RV 2006)*, volume 4262 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2006.

[18] Wolfgang Grieskamp, Nicolas Kicillof, Keith Stobie, and Victor Braberman. Model-based quality assurance of protocol documentation: tools and methodology. *The Journal of Software Testing, Verification and Reliability*, 21(1):55–71, 2011.

[19] Stefan Haar, Claude Jard, and Guy-Vincent Jourdan. Testing input/output partial order automata. In *(TestCom/FATES 2007)*, volume 4581 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 2007.

[20] D. Harel and M. Politi. *Modeling reactive systems with statecharts: the STATEMATE approach.* McGraw-Hill, New York, 1998.

[21] F. C. Hennie. Fault-detecting experiments for sequential circuits. In *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, Princeton, New Jersey, November 1964.

[22] R. M. Hierons. Adaptive testing of a deterministic implementation against a nondetermistic finite state machine. *The Computer Journal*, 41(5):349–355, 1998.

[23] R. M. Hierons. Testing from a non-deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.

[24] Robert M. Hierons. Applying adaptive test cases to nondeterministic implementations. *Information Processing Letters*, 98(2):56–60, 2006.

[25] Robert M. Hierons. Reaching and distinguishing states of distributed systems. *SIAM Journal of Computing*, 39(8):3480–3500, 2010.

[26] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy A. Vilkomir, Martin R. Woodward, and Hussein Zedan. Using formal specifications to support testing. *ACM Computing Surveys*, 41(2), 2009.

[27] Robert M. Hierons and Hasan Ural. Optimizing the length of checking sequences. *IEEE Transactions on Computers*, 55(5):618–629, 2006.

[28] Jiale Huo and Alexandre Petrenko. On testing partially specified IOTS through lossless queues. In *16th IFIP International Conference on the Testing of Communicating Systems (TestCom 2004)*, volume 2978 of *Lecture Notes in Computer Science*, pages 76–94. Springer, 2004.

[29] Jiale Huo and Alexandre Petrenko. Transition covering tests for systems with queues. *Software Testing, Verification and Reliability*, 19(1):55–83, 2009.

[30] Claude Jard, Thierry Jéron, Hakim Kahlouche, and César Viho. Towards automatic distribution of testers for distributed conformance testing. In *TC6 WG6.1 Joint International Conference on Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE 1998)*, volume 135 of *IFIP Conference Proceedings*, pages 353–368. Kluwer, 1998.

[31] Claude Jard, Thierry Jéron, Lénaick Tanguy, and César Viho. Remote testing can be as powerful as local testing. In *Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, volume 156 of *IFIP Conference Proceedings*, pages 25–40. Kluwer, 1999.

[32] D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.

[33] D. Lee and M. Yannakakis. Principles and methods of testing finite-state machines - a survey. *Proceedings of the IEEE*, 84(8):1089–1123, 1996.

[34] G. L. Luo, G. v. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–161, 1994.

[35] A. Jefferson Offutt and Aynur Abdurazik. Generating tests from UML specifications. In *Second International Conference on the The Unified Modeling Language (UML 1999)*, volume 1723 of *Lecture Notes in Computer Science*, pages 416–429. Springer, 1999.

[36] Olaf Owe, Martin Steffen, and Arild B. Torjusen. Model testing asynchronously communicating objects using modulo ac rewriting. *Electronic Notes Theoretical Computer Scence.*, 264(3):69–84, 2010.

[37] A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das. Nondeterministic state machines in protocol conformance testing. In *Proceedings of Protocol Test Systems, VI (C-19)*, pages 363–378, Pau, France, 28-30 September 1994. Elsevier Science (North-Holland).

[38] A. Petrenko, N. Yevtushenko, and G. v. Bochmann. Testing deterministic implementations from nondeterministic FSM specifications. In *Testing of Communicating Systems, IFIP TC6 9th International Workshop on Testing of Communicating Systems*, pages 125–141, Darmstadt, Germany, 9–11 September 1996. Chapman and Hall.

[39] Alexandre Petrenko, Nina Yevtushenko, and Jiale Huo. Testing transition systems with input and output testers. In *15th IFIP International Conference on Testing of Communicating Systems (TestCom 2003)*, volume 2644 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2003.

[40] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[41] John H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, 1984.

[42] Thomas J. Schaefer. The complexity of satisfiability problems. In *Tenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978.

[43] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2), 1972.

[44] P. Tripathy and K. Naik. Generation of adaptive test cases from nondeterministic finite state models. In *Proceedings of the 5th International Workshop on Protocol Test Systems*, pages 309–320, Montreal, September 1992.

[45] Louis Verhaard, Jan Tretmans, Pim Kars, and Ed Brinksma. On asynchronous testing. In *Proceedings of the IFIP TC6/WG6.1 Fifth International Workshop on Protocol Test Systems (Protocol Test Systems V)*, volume C-11 of *IFIP Transactions*, pages 55–66. North-Holland, 1992.

[46] Gregor von Bochmann, Stefan Haar, Claude Jard, and Guy-Vincent Jourdan. Testing systems specified as partial order input/output automata. In *20th IFIP TC 6/WG 6.1 International Conference on Testing of Software and Communicating Systems (TestCom/FATES)*, volume 5047 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2008.

[47] Martin Weiglhofer and Franz Wotawa. Asynchronous input-output conformance testing. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009)*, pages 154–159. IEEE Computer Society, 2009.

[48] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[49] Fan Zhang and To-Yat Cheung. Optimal transfer trees and distinguishing trees for testing observable nondeterministic finite-state machines. *IEEE Transactions on Software Engineering*, 29(1):1–14, 2003.