Implementation relations for testing through asynchronous channels

Robert M. Hierons

Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH
Email: rob.hierons@brunel.ac.uk

This paper concerns testing from an input output transition system (IOTS) model of a system under test that interacts with its environment through asynchronous first in first out (FIFO) channels. It explores methods for analysing an IOTS without modelling the channels. If IOTS M produces sequence σ then, since communications are asynchronous, output can be delayed and so a different sequence might be observed. Thus M defines a language $Tr(M)$ of sequences that can be observed when interacting with M through FIFO channels. We define implementation relations and equivalences in terms of $Tr(M)$: an implementation relation says how IOTS N must relate to IOTS M in order for N to be a correct implementation of M. It is important to use an appropriate implementation relation since otherwise the verdict from a test run might be incorrect and because it influences test generation. It is undecidable whether IOTS N conforms to IOTS M and so also whether there is a test case that can distinguish between two IOTSs. We also investigate the situation in which we have a finite automaton P and either wish to know whether $Tr(M) \cap L(P)$ is empty or whether $Tr(M) \cap Tr(P)$ is empty and prove that these are undecidable. In addition, we give conditions under which conformance and intersection are decidable.

Keywords: Implementation relations; software testing; asynchronous communications; first in first out channels

1. INTRODUCTION

This paper relates to an approach to software testing called model based testing (MBT), in which testing is based on a model of the required behaviour of the system under test (SUT) or some aspect of this. One of the benefits of MBT is that several of the testing activities can be automated, specifically those of generating test cases, running test cases, and returning a verdict (checking whether the behaviour observed was consistent with the model). This has led to significant interest in MBT, with there being industrial evidence of its benefits [1].

An approach to MBT will use an *implementation relation*, which specifies what it means for an SUT to conform to the model M. The implementation relation is required in order to determine the verdict of a test run (whether it is pass or fail) but also influences test generation since it defines what potential behaviours are faulty. An implementation relation is usually defined as a required relationship between two models: the (specification) model M and an unknown model N that models the SUT. It is normal to assume that the SUT can be represented using the modelling notation use for M. Most MBT approaches are based on models expressed as either finite state machines (FSMs) or input output transition systems (IOTSs); while the tester might write a model in a higher-level language, tools will normally convert the model into an FSM or an IOTS. There has therefore been much interest in testing from an FSM or an IOTS (see, for example, [2, 3, 4, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13]).

IOTSs are labelled transition systems in which we distinguish between input and output and are more general than FSMs since, for example, they do not require that input and output alternate. An IOTS is essentially defined by a set of states and transitions between states: each transition is labelled with either an input or an output. The distinction between input and output, made in IOTSs, is important for testing since there is an asymmetry: the tester controls the input and the SUT controls the output. Work on testing from an IOTS often assumes that it is possible to determine whether the SUT is in a state from which it cannot produce an output; this is called quiescence and is represented by δ. It is also normal to assume that the SUT can always receive input (it is input-enabled) and in this context the standard implementation relation is ioco [12].

Most work on testing of state-based systems assumes that these systems interact with their environment through synchronous communication (see, for example,

[14, 1, 12, 13]). This paper is instead concerned with systems that communicate with their environment through asynchronous first in first out (FIFO) channels. This situation is relatively common since many systems interact with users or other systems through networks. In this context there may be a single channel or two channels but we focus on the problem, introduced by such a situation, that there can be a (possibly unpredictable) delay introduced by the network and so the times at which events are observed by the SUT and the test tool are different. An alternative notion of interactions being asynchronous is where the test generation tool cannot process outputs as fast as they are produced by the SUT. This alternative notion of asynchronous interaction causes fewer problems: most test tools have an adapter that connects the tool to the SUT and the adapter can buffer the outputs produced by the SUT. However, such buffers do not overcome problems caused by communicating with an SUT through a network where communications are asynchronous.

There are alternative sources of the problem considered in this paper. For example, the order in which outputs are observed might be affected by the unpredictable behaviour of a thread scheduler. The order of outputs might also not be specified since, for example, the order in which particular values are observed or received at the user interface may not matter. While we focus on the situation in which reordering of events is caused by message latency, it seems likely that the work in this paper can be extended to scenarios such these, possibly by introducing ideas developed in the context of distributed testing (see, for example, [15, 16]).

Asynchronous communications introduces significant challenges since a sequence of events observed need not be that produced by the system: input is received by the system after it has been sent by the environment and output is observed by the environment after it has been produced. Let us suppose, for example, that a system initially sends output $!o$, then receives input $?i$ and sends output $!o'$ in response. Then the system produced the sequence $!o?i!o'$ but, since output can be delayed, the environment might observe $?i!o!o'$. If we use implementation relations defined for synchronous communications and $!o?i!o'$ is a behaviour of the specification and $?i!o!o'$ is not, then the observation of $?i!o!o'$ would lead to the verdict fail even though the SUT behaved in a manner allowed. Note, however, that we assume that communications are FIFO and thus sequence $?i!o'!o$ cannot be observed if the SUT produces $!o?i!o'$ since this would require output $!o'$ to 'overtake' output $!o$.

In this paper we assume that we have a model M of a system, written as an IOTS, and that the interaction of the system with the environment will be through FIFO channels. Most approaches to MBT, where the SUT interacts with the tester through asynchronous channels, augment the specification M with models of the channels (see, for example, [17, 18, 19]), although the problem of deciding whether an observation is consistent with the specification has been investigated [20]. Since the addition of these channels can lead to an infinite state space, or an exponential increase if there are bounded channels, testing is usually on the fly: parts of the state space are only explored when required. However, in deciding whether one IOTS N conforms to another IOTS M we have to explore the entire behaviour of N. We also have to consider the entire behaviour of N if we are looking for test cases that distinguish a potential (faulty) implementation that behaves like N from a specification M. This is highly relevant if we have a fault domain: a set $\mathcal{F}$ of models with the property that the tester believes that the SUT behaves like an unknown member of $\mathcal{F}$. If such a set $\mathcal{F}$ is finite then there is the potential to produce a finite test suite that is guaranteed to determine correctness, subject to the SUT actually behaving like an element of $\mathcal{F}$.

There has been work that has defined equivalences for systems that interact with their environment through FIFO channels and that does so without introducing models of the channels [21, 22, 23]. Three equivalences have been studied [21]. One equivalence, $\sim_Q$ corresponds to some approaches that compose the IOTS with models of the queues. One potential problem with $\sim_Q$ is that it requires the environment to be able to determine when the system is quiescent (when it cannot change state or produce output without receiving input) and this may be problematic when communicating through asynchronous channels unless we have information regarding message latency. In a second equivalence, $\sim_{io}$, an input sequence is applied and the resultant output sequence is recorded but the overall sequence of inputs and outputs is not observed. Thus, $\sim_{io}$ is a weak equivalence: it would not note, for example, that the observation of output $!o$ before input $?i$ is not consistent with a specification in which $!o$ should be produced after $?i$. It transpires that both of these equivalences are undecidable [21]. Under a third equivalence, $\sim_{ioblock}$, sequences of pairs of input and output sequences are observed. While this is a stronger equivalence than $\sim_{io}$, we will see that there is still some loss of relevant information regarding the order in which inputs and outputs are observed. In addition, both $\sim_{io}$ and $\sim_{ioblock}$ make observations in quiescent states: this both means that there is an implicit requirement to be able to observe quiescence but also that these equivalences may not be suitable when considering systems that can fail to reach a quiescent state. Simple examples of systems that can fail to reach a quiescent state are a clock that outputs 'tick' repeatedly and a screen saver. In this paper we define new implementation relations and equivalences that do not require us to observe quiescence.

In this paper we define the language $Tr(N)$ for an

IOTS N. This is the set of sequences that can be formed from sequences of N through 'delaying' output and thus through sequences of transformations of the form $!o?i \rightarrow ?i!o$ for input $?i$ and output $!o$. Thus, $Tr(N)$ models the fact that an output is observed after it is sent by the process N being considered. This leads to the implementation relation under which N conforms to M if and only if $Tr(N) \subseteq Tr(M)$. However, it is also possible that a sequence observed is formed from a sequence of N through two additional factors: input has been sent (and so observed by the environment) but has not been received by N and output has been sent by N but has not yet been observed by the environment. We let $Tr^p(N)$ denote the set of such sequences. The sequences in $Tr^p(N)$ that are not in $Tr(N)$ can be seen as being partial observations. However, unless we have bounds on the latency caused by message transmission, in observing a system we cannot know whether an observed sequence from $Tr^p(N)$ is in $Tr(N)$. Thus, we obtain an addition implementation relation under which N conforms to M if and only if $Tr^p(N) \subseteq Tr^p(M)$. Interestingly, it transpires that these two implementation relations are equivalent: $Tr^p(N) \subseteq Tr^p(M)$ if and only if $Tr(N) \subseteq Tr(M)$.

Having defined an implementation relation, we investigate two problems. First, we consider the situation in which M is a specification of how a system should behave, we have another IOTS model N, and we want to know whether N is a suitable implementation or design of M. This reduces to determining whether $Tr(N) \subseteq Tr(M)$. In the context of software testing, this problem also corresponds to that of asking whether there is any test case that is capable of distinguishing between N and M when communicating through FIFO channels, a problem that is particularly relevant if M is a specification and N models a potential (possibly faulty) implementation. We prove that the problem of determining whether $Tr(N) \subseteq Tr(M)$ is generally undecidable but that it is decidable in low-order polynomial time for alternating IOTSs, a type of model and system that has been widely studied (see, for example, [24, 25, 26]).

The result, that $Tr(N) \subseteq Tr(M)$ is generally undecidable, has consequences for testing as well as verification. One approach to reasoning about test effectiveness is to use a fault domain $\mathcal{F}$, which is a set of models that describes the possible behaviours of the system under test (SUT) (see, for example, [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]). Given such a fault domain $\mathcal{F}$ and a test suite $\mathcal{T}$, we can ask whether $\mathcal{T}$ distinguishes all faulty elements of $\mathcal{F}$ from M. Fault domains have received particular attention in testing from an FSM, the standard fault domains for FSMs placing upper bounds on the number of states of the (unknown) FSM that represents the SUT. As originally noted by Moore [11], it is then possible to produce a test suite that distinguishes each faulty element of $\mathcal{F}$ from the specification, since conformance is decidable and $\mathcal{F}$ is finite. While it generally is not feasible to produce test suites by separately considering the elements of the fault domain, this observation led to the development of techniques for generating test suites with guaranteed fault detection (see, for example, [27, 3, 4, 5, 9, 10]). The result regarding $Tr(N) \subseteq Tr(M)$ suggests that it will be difficult to extend this approach to testing from an IOTS when there are asynchronous FIFO communications since it is undecidable whether an element of a fault domain conforms to the specification.

We can represent the problem of deciding whether $Tr(N) \subseteq Tr(M)$ in terms of semi-commutations. A semi-commutation is defined by a possibly asymmetric independence relation $\mathcal{I}$ over the alphabet Σ. Having $(a, b) \in \mathcal{I}$ denotes the possibility of rewriting ab to form ba. Semi-commutations are a generalisation of partial commutations, in which $\mathcal{I}$ is required to be symmetric. Since inclusion is known to be undecidable for rational partial commutations[1], it is also undecidable for rational semi-commutations. However, the problem we investigate in this paper has an anti-symmetric independence relation $\mathcal{I} = O \times I$ in which I and O partition the alphabet and so is a class of semi-commutations in which $\mathcal{I}$ cannot be symmetric. Thus, decidability results regarding partial commutations do not apply. In proving that the implementation relations are undecidable we therefore also prove that inclusion is undecidable for rational semi-commutations where the independence relation is of the form $\Sigma_1 \times \Sigma_2$ for Σ_1 and Σ_2 that partition the alphabet Σ.

In addition to conformance, we consider the situation in which we have an IOTS model M and a finite automaton P that defines a property of interest. We assume that we wish to know whether there is a sequence of observations that can be made when interacting with M through FIFO channels that can also be made when interacting with P through FIFO channels. Thus, we wish to know whether $Tr(M) \cap Tr(P) = \emptyset$. This can be seen as a problem of deciding whether there are observations that can be made of M that are consistent with P, a problem that might be of interest if P models some undesirable behaviours and we wish to know whether the observations regarding M can be consistent with such behaviours. Similar problems have been considered for message sequence charts [28, 29]. We then consider the situation in which we wish to know whether there is any sequence of observations, that can be made when interacting with M, that is in the language defined by P. Thus, we wish to know whether $Tr(M) \cap L(P) = \emptyset$, where $L(P)$ denotes the set of sequences in the regular language defined by P. In this situation, the sequences in $L(P)$ might correspond to sequences that are undesirable since, for example, M will interact through FIFO

[1] A rational partial commutation is defined by a finite automaton A and an independence relation $\mathcal{I}$. The language defined by this is the set of sequences that can be formed from sequences in $L(A)$ through transformations based on by $\mathcal{I}$.

channels with another system M' and M' should not receive such sequences. We prove that these problems are generally undecidable but can be solved in low-order polynomial time if the models are alternating.

This paper makes the following contributions. First, it defines new implementation relations and equivalences for IOTSs that model systems that interact with their environment through asynchronous FIFO channels. Unlike previous approaches, these implementation relations and equivalences do not require the observation of quiescence and do not require us to explicitly model the communications channels. It also proves that conformance and equivalence are generally undecidable but can be decided in low-order polynomial time for alternating models. An additional consequence of the results is that it is undecidable whether there is a test case that distinguishes two IOTSs M and N when communicating through asynchronous FIFO channels. In proving these results we also proved that inclusion and equivalence are undecidable for rational semi-commutations in which the independence relation is of the form $\Sigma_1 \times \Sigma_2$ where Σ_1 and Σ_2 partition the alphabet. Given finite automaton P we also consider the problems of deciding whether $Tr(M) \cap L(P)$ is empty and whether $Tr(M) \cap Tr(P)$ is empty, proving that these are undecidable in general but can be solved in low-order polynomial time for alternating models.

The paper is structured as follows. Section 2 describes preliminary material. In Section 3 we define implementation relations and in Section 4 we explore the decidability of these: we prove that they are generally undecidable but are decidable in low-order polynomial time for alternating models. Section 5 then considers the problem where we want to know whether any elements of $Tr(M)$ are in a language defined by a finite automaton and proves that the two forms of this problem are undecidable. Finally, Section 6 draws conclusions and discusses possible future work.

2. PRELIMINARIES

In this section we start by describing IOTSs and give associated definitions and we then move on to discuss asynchronous communications.

2.1. Input Output Transition Systems

We will be concerned with systems that interact with their environment through asynchronous FIFO channels. Throughout the paper we use I to denote the input alphabet of the system and O to denote its output alphabet. We let $Act = I \cup O$ denote the set of events. We use the normal convention in which the name of an input is preceded by ? and the name of an output is preceded by !.

An Input Output Transition System (IOTS) is defined by a set of states with transitions between them, with each transition having a label that is either an input or an output[2]. We restrict attention to IOTSs that have finite sets of states, inputs and outputs since we are concerned with decidability and complexity issues.

Definition 2.1. *An IOTS M is 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 $S \times Act \leftrightarrow S$. For $s \in S$ and $a \in Act$, $h(s, a)$ denotes the set of $s' \in S$ such that (s, a) and s' are related under h. If $s' \in h(s, a)$ then M can move from state s to state s' through $a \in Act$ and this defines the* transition (s, a, s') *with* starting state s, ending state s', *and* label a. *A transition with the same starting and ending states is a* self-loop transition. *An IOTS is* input-enabled *if for every state and input there is at least one associated transition: for all $s \in S$ and $?i \in I$, $h(s, ?i) \neq \emptyset$.*

Equivalences have been defined for IOTSs that are not input-enabled and communications is asynchronous [21] but these correspond to defining equivalences on input-enabled IOTSs formed by completing the process in the following way: add an error state s_e and self-loops labelled with inputs in s_e and for each $(s, a) \in S \times Act$ such that $h(s, a) = \emptyset$, add the transition (s, a, s_e). Thus, we can instead transform IOTSs to make them input-enabled and define equivalences on the resultant IOTSs. Since this process adds only one state to an IOTS, and considering input-enabled IOTSs simplifies the exposition, we assume that all IOTSs considered are input-enabled.

Definition 2.2. *Given an IOTS M, a path $\rho = (s_1, a_1, s_2) \dots (s_k, a_k, s_{k+1})$ is a sequence of consecutive transitions. We say that s_1 is the* starting state *of ρ, s_{k+1} is the* ending state *of ρ, and $a_1 \dots a_k$ is the* label *of ρ. Given IOTS M we let $L(M)$ denote the set of labels of paths of M that have starting state s_0. We use ϵ to represent an empty sequence. A path is a* cycle *if it has the same starting and ending states.*

Work on testing from an IOTS through synchronous channels often assumes that we can observe quiescence: the situation in which the system cannot progress (change state and/or produce output) without receiving additional input [12]. However, this may be problematic when interacting with a system through asynchronous channels, since message latency can be unpredictable, and so we do not include quiescence in our set of observations.

We will also consider finite automata (FA).

Definition 2.3. *A FA P is defined by a tuple (S, s_0, A, h, F) in which S is a finite set of states; $s_0 \in S$*

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

is the initial state; A is the finite alphabet; h is the transition relation of type $S \times A \leftrightarrow S$; and $F \subseteq S$ is the set of final states. Given FA P, we let $L(P)$ denote the set of labels of paths of P that have starting state s_0 and ending state in F.

Since FA can be considered to be IOTS in which we have a set of final states and we do not differentiate between input and output, we reuse notation and terms defined for IOTSs when discussing FA.

Finally, we will consider a special type of model in which input and output alternate, a condition that has received some attention [24, 25, 26]. In an alternating IOTS (AIOTS) the state set S is partitioned into subsets S_I and S_O such that the initial state is in S_I, all transitions from states in S_I are labelled with inputs and reach states in S_O, and all transitions from states in S_O are labelled with outputs and reach states in S_I. Clearly, an AIOTS cannot be input-enabled as defined for IOTSs. Thus, we say that an AIOTS is input-enabled if all inputs are possible in each input state and we assume that any AIOTS considered is input-enabled. We can similarly define alternating FA.

2.2. Asynchronous observation and semi-commutation

In interacting through asynchronous (FIFO) channels with a system, that behaves like an IOTS M, the sequence σ observed need not be one from $L(M)$ since output is observed by the environment after it is produced by the system. Thus, a sequence σ might be observed if there is some $\sigma' \in L(M)$ such that we can produce σ from σ' by delaying output. The delaying of output can be represented by transformations of the form $\sigma_1 !o ?i \sigma_2 \rightarrow \sigma_1 ?i !o \sigma_2$ for input $?i$ and output $!o$ and we can produce σ from σ' by delaying output if we can transform σ' into σ using a sequence of such transformation steps.

EXAMPLE 1. Consider sequence $\sigma = !o?i!o$ produced by the SUT. Since output can be delayed, $!o$ might not be observed until after $?i$ is sent by the tester and so the tester might observe the sequence $\sigma' = ?i!o!o$. By definition, $\sigma \rightarrow \sigma'$. However, this is the only sequence other than σ that can be observed in interacting with an SUT through FIFO channels if σ is produced by the SUT and so, for example, the observation $!o!o?i$ could not occur if the SUT produces σ.

Given an alphabet Σ, a *partial commutation* is defined by a symmetric and anti-reflexive *independence relation* $\mathcal{I} \in \Sigma \times \Sigma$, where $(a, b) \in \mathcal{I}$ implies that ab and ba are equivalent. The complement of $\mathcal{I}$, $D = (\Sigma \times \Sigma) \setminus \mathcal{I}$, is the *dependence relation*. The independence relation defines an equivalence relation on sequences: σ and σ' from Σ^* are equivalent given $\mathcal{I}$ if we can transform σ into σ' using a sequence of transformations of the form $\sigma_1 ab \sigma_2 \rightarrow \sigma_1 ba \sigma_2$ for $(a, b) \in \mathcal{I}$. An equivalence class under this relation is said to be a trace and the study of such traces is often called Mazurkiewicz trace theory [30, 31].

When considering asynchronous communication, the notion of independence is not symmetric since we have rules of the form $\sigma_1 !o ?i \sigma_2 \rightarrow \sigma_1 ?i !o \sigma_2$ but no rules of the form $\sigma_1 ?i !o \sigma_2 \rightarrow \sigma_1 !o ?i \sigma_2$. It therefore corresponds to a *semi-commutations* rather than a partial commutation, a semi-commutation having an independence relation $\mathcal{I}$ that need not be symmetric [32]. As before, given a sequence $\sigma \in \Sigma$, we have a class $[\sigma\rangle$ of sequences that can be formed through applying transformations defined by $\mathcal{I}$ to σ. However, since $\mathcal{I}$ need not be symmetric, $[\sigma\rangle$ need not be an equivalence class.

EXAMPLE 2. Consider the sequence $\sigma = !o!o?i!o$ produced by the SUT and the sequence $\sigma' = ?i!o!o!o$. It is straightforward to see that σ cannot be transformed into σ' through one transformation of the form $\sigma_1 !o ?i \sigma_2 \rightarrow \sigma_1 ?i !o \sigma_2$. However, $!o!o?i!o \rightarrow !o?i!o!o \rightarrow ?i!o!o!o$ and so $\sigma' \in [\sigma\rangle$. In contrast, $\sigma \notin [\sigma'\rangle$.

We can now express the set of observations, that can be made when communicating with IOTS M through asynchronous FIFO channels, in terms of a semi-commutation. This semi-commutation has independence relation $\mathcal{I} = O \times I$, since output can be delayed but the channels are FIFO, and dependence relation $D = (I \times O) \cup (I \times I) \cup (O \times O)$. Note also that this is a restriction on the notion of semi-commutations since $\mathcal{I}$ is anti-symmetric and not just asymmetric and also because I and O partition Act.

DEFINITION 2.4. *Given IOTS M we let $Tr(M)$ denote the set of sequences that are in classes defined by elements of $L(M)$. Thus, $Tr(M) = \bigcup_{\sigma \in L(M)} [\sigma\rangle$. This is the set of sequences that can be formed from sequences in $L(M)$ through delaying output. If $\sigma' \in [\sigma\rangle$ then we write $\sigma \rightsquigarrow \sigma'$.*

Implementation relations will be defined in terms of $Tr(M)$.

3. IMPLEMENTATION RELATIONS AND EQUIVALENCES

In this section we explore possible implementation relations for testing from an IOTS model when we communicate with the SUT through asynchronous channels. In Section 4 we explore the problem of deciding whether one model conforms to another under such an implementation relation. Recall that an implementation relation defines the situations in which one model conforms to another and is required in order to determine the verdict of testing: the use of the wrong implementation relation could either lead to failures being missed (verdict fail should be produced but is not) or unsound testing (a fail verdict is produced even though the behaviour observed should be one allowed). The implementation relation used also influences test generation since it defines what constitutes a faulty

implementation.

Recent work has discussed three equivalences for asynchronous testing [21]. One equivalence, $\sim_Q$, corresponds to some approaches that compose the IOTS with models of the queues. However, $\sim_Q$ requires the environment to be able to observe quiescence. Since we do not make this assumption, $\sim_Q$ is not suitable for our purposes. In a second equivalence, $\sim_{io}$, an observation is a pair (u, v) in which u is an input sequence and v is an output sequence: the overall sequence of inputs and outputs is not considered. It is required that the sequences u and v are projections of a sequence from Act^* that takes the process to a quiescent state but otherwise there is no need to observe quiescence. As noted earlier, $\sim_{io}$ is a weak equivalence since it would consider $!o?i$ to be acceptable if $?i!o$ is a behaviour of the specification even if $!o?i$ is not. The observation of separate sequences of inputs and outputs is consistent with some work on testing [33] but typically we instead observe a sequence of inputs and outputs since this can provide additional information.

Under a third equivalence, $\sim_{ioblock}$, sequences of pairs of input and output sequences are observed. When considering IOTS M, such a sequence $(u_1, v_1) \dots (u_k, v_k)$ must have the property that there is a sequence $\sigma = \sigma_1 \dots \sigma_k \in L(M)$ such that σ labels a path ρ of M from the initial state of M where for all $1 \leq i \leq k$ we have that u_i and v_i are projections of σ_i onto I and O respectively and also that the prefix of ρ with label $\sigma_1 \dots \sigma_i$ reaches a quiescent state. Again there is a potential loss of information since this may not give all of the relevant information regarding the order in which inputs and outputs are observed (see Example 3 below). It transpires that $\sim_{ioblock}$ is decidable when considering well-formed IOTSs (WIOTSs) [21]. An IOTS is a WIOTS if the set S of states can be partitioned into a set of input states (S_I) and a set of output states (S_O). All transitions from states in S_I are labelled with inputs and all transitions from states in S_O are labelled with outputs. However, it was left open whether $\sim_{ioblock}$ is decidable in general. Since the previously defined equivalences all have some limitations, and all either explicitly or implicitly require the observation of quiescence, we now explore implementation relations and equivalences based on semi-commutations. Similar to $\sim_{io}$ and $\sim_{ioblock}$ we define these in terms of IOTSs without composing them with models of queues. In this paper we assume that sequences of inputs and outputs are observed. The following shows that $\sim_{io}$ and $\sim_{ioblock}$ are too weak.

Example 3. Consider the IOTSs M_1 and N_1 shown in Figure 1. Here states are represented by circles, with the initial states being represented by the circles at the top, and an arc with label a represents a transition with label a. For each IOTS there are only two quiescent states: the states where all outgoing transitions are self-loops. An observation under $\sim_{io}$ is a pair containing an input sequence and an output sequence. Thus, for example, the observations of N_1 include pairs such as $(?i, !o!o!o)$ and $(\epsilon, !o!o!o)$ but $(\epsilon, !o)$ is not an observation since there is no path of N_1 that reaches a quiescent state and has these projections. Both IOTSs give two types of possible observation under $\sim_{io}$ and $\sim_{ioblock}$: the pairs $(\epsilon, !o!o!o)$ and $(?i, !o!o!o')$ with arbitrarily many instances of $?i$ added due to the self-loops. Thus, M_1 and N_1 are equivalent under $\sim_{io}$ and $\sim_{ioblock}$. However, if M_1 is the specification then there is a possible observation that can be made regarding N_1 that cannot be made with M_1: the sequence $!o!o?i!o'$. While $!o?i!o!o' \in L(M)$, the sequence $!o!o?i!o'$ cannot be observed when interacting with M_1 since this would require an input to be delayed or, equivalently, for an output to be observed before it is produced. This suggests that N_1 should not be considered to be a good implementation of M_1 and thus that $\sim_{io}$ and $\sim_{ioblock}$ are too weak.

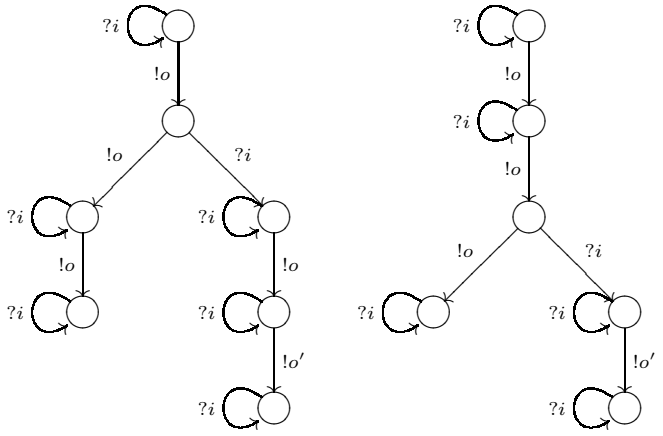

FIGURE 1. IOTSs M_1 and N_1

However, the use of asynchronous communications does affect our ability to distinguish between IOTSs in testing.

Example 4. Consider again the specification M_1 in Figure 1 but now let us suppose that the SUT behaves like N_1' given in Figure 2. It is clear that N_1' does not conform to M_1 if testing is synchronous since we can observe $\sigma = ?i!o!o!o'$ when interacting with N_1' and this is not in $L(M_1)$. However, σ and its extensions by input sequences are the only elements of $L(N_1')$ that are not in $L(M_1)$ and σ can be formed by delaying the first output in $\sigma' = !o?i!o!o' \in L(M_1)$. Thus, in asynchronous testing we cannot distinguish N_1' from M_1.

As discussed earlier, the other problem regarding requiring observations to be made in quiescent states is that there may be behaviours that are not prefixes of observations that end in quiescence and such behaviours are not considered.

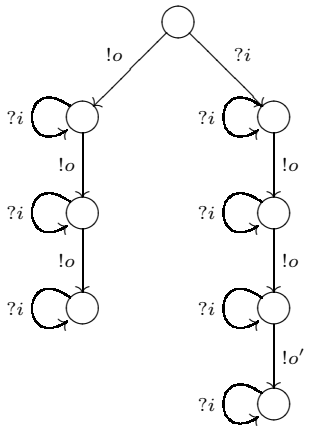

FIGURE 2. IOTS N_1' that conforms to M_1

EXAMPLE 5. Consider processes M_2 and N_2 defined by the following:

- M_2 has only one state, in which there are self-loop transitions labelled with all of the inputs and also the output $!o$.
- N_2 has only one state, in which there are self-loop transitions labelled with all of the inputs and also the output $!o' \neq !o$.

Clearly, N_2 can produce the output $!o'$ and M_2 cannot and so N_2 should not be considered to be a correct implementation of M_2. However, M_2 and N_2 have no quiescent states and so are equivalent under both $\sim_{io}$ and $\sim_{ioblock}$.

In order to define alternative implementation relations and equivalences we need to consider the possible observations that might be made. This leads to the following implementation relation and equivalence.

DEFINITION 3.1. *Given IOTSs N and M with the same alphabets, we say that $N \sqsubseteq M$, if $Tr(N) \subseteq Tr(M)$. Further, $N \equiv M$ if $Tr(N) = Tr(M)$.*

In addition, if $Tr(N)$ is the set of observations that might be made of N then it is possible to distinguish N from M in testing through asynchronous FIFO channels if and only if $Tr(N) \not\subseteq Tr(M)$. The idea here is straightforward: the sequence σ can distinguish N from M if σ is an observation that can be made regarding N ($\sigma \in Tr(N)$) and σ is not an observation that can be made regarding M ($\sigma \notin Tr(M)$).

A potential weakness of these relations is that we might not know whether all of the inputs sent to the system have been received and also whether all of the outputs that have been produced have been observed. One possible solution is to require that observations are made in quiescent states only; an approach used for $\sim_{io}$ and $\sim_{ioblock}$. This leads to the following in which for IOTS M we have that $L_\delta(M)$ denotes the set of labels of paths of M that end in quiescent states and $Tr_\delta(M)$ denotes the set of sequences that are either in $L_\delta(M)$ or can be produced from a sequence in $L_\delta(M)$ through delaying output ($Tr_\delta(M) = \cup_{\sigma \in L_\delta(M)}[\sigma\rangle$). While this requires us to be able to determine when the system is quiescent, it is included for the sake of completeness.

DEFINITION 3.2. *Given IOTSs N and M with the same alphabets, we say that $N \sqsubseteq_\delta M$, if $Tr_\delta(N) \subseteq Tr_\delta(M)$. Further, $N \equiv_\delta M$ if $Tr_\delta(N) = Tr_\delta(M)$.*

A final alternative is, for a process M, to define a language that represents sequences that might be observed when interacting with M through FIFO channels and where some messages may not have arrived yet. Here, not only might output be delayed but observed inputs might not have been received by M yet and outputs produced by M might not have been observed yet. Thus, this language can be seen as representing possible 'partial' behaviours of M that might be observed: without bounds on message latency we could have to wait arbitrarily long to observe further output. This leads to the following definition of $Tr^p(M)$, which is the set of sequences that can be formed from sequences in $L(M)$ through transformations that either add input (that has not been received yet), delete output (since it might not have been observed yet), or delay output.

DEFINITION 3.3. *Given IOTS M, $Tr^p(M)$ denotes the set of sequences that can be formed from sequences in $L(M)$ through zero or more transformations of the following types.*

- $\sigma_1 !o ?i \sigma_2 \rightarrow \sigma_1 ?i !o \sigma_2$ *for input* $?i$ *and output* $!o$.
- $\sigma \rightarrow \sigma ?i$ *for input* $?i$.
- $\sigma_1 !o \sigma_2 \rightarrow \sigma_1 \sigma_2$ *for output* $!o$ *and sequence* $\sigma_2 \in I^*$.

If σ' can be formed from σ through such transformations then we write $\sigma \rightsquigarrow^p \sigma'$.

EXAMPLE 6. Consider the sequence $\sigma = !o?i!o$. Then clearly $\sigma \rightsquigarrow^p ?i!o!o$ since it is possible to delay output under $\rightsquigarrow^p$. However, under $\rightsquigarrow^p$ we can also add input (that has not been received by the SUT) and so we also have that $\sigma \rightsquigarrow^p !o?i!o?i$. Finally, we can have failed to observe some of the output produced and so $\sigma \rightsquigarrow^p !o?i$ and also $\sigma \rightsquigarrow^p ?i$.

Note that it is sufficient to add an input to the end of a sequence since further transformations can then move outputs past it. We can now define an implementation relation and an equivalence based on $Tr^p(M)$.

DEFINITION 3.4. *Given IOTSs N and M with the same alphabets, we say that $N \sqsubseteq^p M$, if $Tr^p(N) \subseteq Tr^p(M)$. Further, $N \equiv^p M$ if $Tr^p(N) = Tr^p(M)$.*

However, implementation relations $\sqsubseteq$ and $\sqsubseteq^p$ are equivalent.

PROPOSITION 3.1. *Given IOTSs N and M, $N \sqsubseteq M$*

if and only if $N \sqsubseteq^p M$.

Proof. First assume that $N \sqsubseteq M$ and we are required to prove that $N \sqsubseteq^p M$. Let σ be some sequence in $Tr^p(N)$ and it is sufficient to prove that $\sigma \in Tr^p(M)$. Since $\sigma \in Tr^p(N)$ there is some sequence $\sigma' \in L(N)$ such that $\sigma' \rightsquigarrow^p \sigma$. But, since $\sigma' \in L(N)$ we have that $\sigma' \in Tr(N)$ and so, since $N \sqsubseteq M$, $\sigma' \in Tr(M)$. Thus, since $\sigma' \rightsquigarrow^p \sigma$, $\sigma \in Tr^p(M)$ as required.

It is therefore sufficient to prove that $N \sqsubseteq^p M$ implies that $N \sqsubseteq M$. We assume that $N \sqsubseteq^p M$ and let σ denote some element of $Tr(N)$: it is sufficient to prove that $\sigma \in Tr(M)$.

Since $N \sqsubseteq^p M$, $\sigma \in Tr^p(M)$ and so there is some sequence $\sigma_1 \in L(M)$ such that $\sigma_1 \rightsquigarrow^p \sigma$. Let σ_2 denote the longest prefix of σ_1 such that σ_2 and σ have the same projections onto O and assume that σ has k outputs. Since $L(M)$ is prefix closed, $\sigma_2 \in L(M)$. Now define the sequence σ_3 that has the same projection onto I as σ and that is formed from σ_2 by adding inputs to the end (σ_2 is a prefix of σ_3). Since M is input-enabled, $\sigma_3 \in L(M)$.

By construction, σ_3 and σ have the same input and output projections. In addition, let us suppose that σ has prefix $\sigma'!o$ and σ_3 has prefix $\sigma_3'!o$ where both σ' and σ_3' contain exactly $i-1$ outputs. Then, since σ_3 is formed by adding inputs to the end of σ_2, $\sigma_3'!o$ is a prefix of σ_2 and so also of σ_1. Further, we know that $\sigma_1 \rightsquigarrow^p \sigma$. Thus, the input projection of σ_3' must be a prefix of the input projection of σ'.

To conclude, σ_3 and σ have the same input and output projections and for all $1 \leq i \leq k$, the sequence of inputs before the ith output in σ_3 is a prefix of the sequence of inputs before the ith output in σ. Thus, $\sigma_3 \rightsquigarrow \sigma$ and so $\sigma \in Tr(M)$ as required. □

4. DETERMINING CONFORMANCE

Section 3 defined three implementation relations, $\sqsubseteq$, $\sqsubseteq_\delta$, and $\sqsubseteq^p$ for testing through asynchronous channels and proved that $\sqsubseteq$ and $\sqsubseteq^p$ are equivalent. In testing we might want to determine whether two models M and N are related under these implementation relations, possibly because M is the model from which we are testing and N is a possible model of the SUT. We would like to determine whether N would be a correct model of the SUT; if it is not we would then like to produce a test that can demonstrate that an SUT behaving like N is faulty. In this section we therefore explore the problem of deciding whether implementation relations $\sqsubseteq$ and $\sqsubseteq_\delta$ and equivalences $\equiv_\delta$ and $\equiv$ are decidable. In Section 4.1 we prove that these are not decidable and in Section 4.2 we give conditions under which they are decidable. Since $\sqsubseteq^p$ and $\sqsubseteq$ are equivalent, we use $Tr(N)$ but not $Tr^p(N)$ when exploring observations that might be made regarding IOTS N.

4.1. The general problem

It is known that the problem of deciding inclusion for rational partial commutations[3] is undecidable (see, for example, [30]). As a result, this problem is also undecidable for rational semi-commutations, which generalise partial commutations by not requiring that $\mathcal{I}$ is symmetric. This may explain why there appears to be no results regarding deciding language inclusion for rational semi-commutations. However, we are interested in a particular type of semi-commutation, in which $\mathcal{I} = O \times I$ and I, O partition Act. As explained earlier, this class of semi-commutations does not include the partial commutations, since $\mathcal{I}$ is anti-symmetric. Thus, results regarding partial commutations do not extend to such semi-commutations. Note that an atomic semi-commutation [34, 35] has an anti-symmetric independence relation. However, for an atomic semi-commutation, while $\mathcal{I}$ must be in the form of $\Sigma_1 \times \Sigma_2$ for disjoint Σ_1 and Σ_2, these sets need not partition Act. The problem of deciding inclusion for atomic semi-commutations also appears not to have been studied. We now show that it is generally undecidable whether $Tr(N) \subseteq Tr(M)$, and thus also that inclusion and equivalence are undecidable for rational semi-commutations of the type described above.

We first define an IOTS M_T that has a particular relationship with a given Turing Machine T that halts if it reaches the halt state with an empty tape. We will define M_T and another IOTS M such that $M_T \sqsubseteq M$ if and only if the Turing machine T does not halt on the empty tape. Since the halting problem is undecidable, this show that it is generally undecidable whether $N \sqsubseteq M$ for IOTSs N and M: if $N \sqsubseteq M$ was decidable then we could use the corresponding algorithm to solve the halting problem. The construction of M_T is similar to one used in Floyd's proof that Post's Correspondence Problem is undecidable [36], as described by Davis and Weyuker [37]. For the set Σ described below, which includes the alphabet of T, M_T will have an input set that contains a copy Σ_I of Σ and an output set that contains a copy Σ_O of Σ. We construct M_T so that there is a particular type of non-empty sequence σ in $Tr(M_T)$ with the same input and output projections ρ (once decorating subscripts are removed) if and only if ρ defines a halting computation of T when started with an empty tape.

Let Σ denote the set formed by adding to the alphabet of T the name of each state of T plus special symbols #, γ, and Δ, with # being used to separate configurations of T, γ being used to denote the end of a computation, and Δ representing a blank cell of the tape. We use two copies of Σ, called Σ_I and Σ_O, in which each element is decorated with a subscript of

[3] A rational partial commutation is a language defined by a finite automaton and a symmetric independence relation on the alphabet.

I (for Σ_I) or O (for Σ_O). Given $\sigma \in \Sigma^*$ we let σ_I and σ_O denote the sequences formed from σ by adding subscripts I and O respectively to its letters. The IOTS M_T will have alphabet $\Sigma_C = \Sigma_I \cup \Sigma_O$ in which Σ_I is the set of inputs and Σ_O the set of outputs.

In constructing M_T we represent possible configurations of T using sequences from Σ_I^* or Σ_O^*. A configuration of a Turing machine is defined by the tape contents, the state, and the current location of the tape head. We use the position of the state name, in a sequence defining a configuration, to indicate the location of the head of T. Thus, a sequence of the form $\sigma_1 q \sigma_2$ represents a configuration in which the current state is q, the tape contains $\sigma_1\sigma_2$ and the head of T is currently on the cell represented by the first element of σ_2. Configurations are separated by copies of #.

M_T will be formed so that its paths start with an input sequence that represents the initial configuration of T (followed by $\#_I$). We will be interested in paths that have the same input and output projections and these will represent possible sequences of configurations for T. After the initial path, containing input that represents the initial configuration of T, M_T can loop in a state s. In order for the input and output projections of the label of a path to be the same, M_T must follow this prefix by a path whose output projection starts with a representation of the initial configuration and we define M_T so that in forming this it must include input that represents a configuration that can be reached from the initial configuration.

The loops in state s represent possible changes in configuration by them either having label $a_I a_O$ for some $a \in \Sigma$ (representing a part of the configuration that has not changed) or representing a change in part of the configuration. The latter will start with a representation of the 'new' part of the configuration using inputs and follow this by a representation of the 'old' part of the configuration using outputs. Importantly, changes in the configuration are limited: the most that can happen is that the contents of the cell being read changes and the tape head moves one place. Let us suppose, for example, that T can move from q to q' and move the head of the tape to the right if the head is reading a and that this does not alter the contents of the cell. This is represented by a loop with label $a_I q'_I q_O a_O$. All such changes in configuration can be represented by a finite set of such loops. Finally, M_T can follow a path from s whose label is output that represents the halting configuration of T (T halting with an empty tape) and this is followed by a path with label $\gamma_I\gamma_O$.

Now let us suppose that M_T has a path ρ whose label is of the form $\sigma\gamma_I\gamma_O$ and has the same projections onto Σ_I and Σ_O (once decorating subscripts are removed). Then these projections of σ must both start with a sequence that represents the initial configuration of T and they must both end with sequences that represent the unique halting configuration. Now consider the suffix σ_1 of σ that follows the sequence of inputs representing the initial configuration. Then the projection onto Σ_O must start with the initial configuration. In addition, assuming this initial configuration is not the halting configuration, this is formed by self-loops in state s. By construction, the projection of σ_1 onto Σ_I must start with a configuration that can be reached from the initial configuration through one transition. We can now repeat this reasoning, until finally the halting configurations are included. Thus, the projections of σ onto Σ_I and Σ_O must represent a computation of T that starts with the initial configuration and ends with the halting configuration. Finally, we complete M_T by, for each state s' and input $?i$ such that there is no transition from s with label $?i$, adding a transition $(s', ?i, s_e)$ to a new state s_e from which there are only self-loop transitions whose labels are inputs.

The IOTS M_T has some important properties that are immediate from its definition. The first shows how the language $L(M_T)$ relates to halting computations of T on an empty tape.

LEMMA 4.1. *Given Turing Machine T, there is a sequence σ in $L(M_T)$ that ends in $\gamma_I\gamma_O$ and has input and output projections $\rho_I\gamma_I$ and $\rho_O\gamma_O$ for some $\rho \in \Sigma^*$ if and only if ρ defines a halting computation of T when started with an empty tape.*

The following will be used to reason about the set of elements in $Tr(M_T)$ that are not in $L(M_T)$.

LEMMA 4.2. *Given Turing Machine T and IOTS M_T, every sequence in $L(M_T)$ that does not reach s_e has at least as many inputs as outputs.*

We now use these results to prove that conformance is undecidable for the implementation relations $\sqsubseteq$ and $\sqsubseteq_\delta$ by showing that for the halting problem for a Turing machine T can be represented in terms of deciding $\sqsubseteq$ or $\sqsubseteq_\delta$.

THEOREM 4.1. *Let us suppose that M and N are IOTSs with the same input and output alphabets. Then the following are undecidable*

1. $N \sqsubseteq M$.
2. $N \sqsubseteq_\delta M$.

Proof. First consider the problem of deciding whether $N \sqsubseteq M$ and so whether $Tr(N) \subseteq Tr(M)$. We will assume that we have been given a Turing machine T and will use the IOTS M_T defined above. We will define an IOTS M such that $Tr(M_T) \not\subseteq Tr(M)$ if and only if T halts on the empty tape.

We now define an IOTS M with input and output alphabets Σ_I and Σ_O respectively. This IOTS is shown in Figure 3 in which a_I denotes all elements from $\Sigma_I \setminus \{\gamma_I\}$ and a_O denotes all element from $\Sigma_O \setminus \{\gamma_O\}$. Further, $a_I b_O$ denotes sequences of the form $a_I b_O$ in which $a \neq b$ and $a_I a_O$ denotes such sequences in which

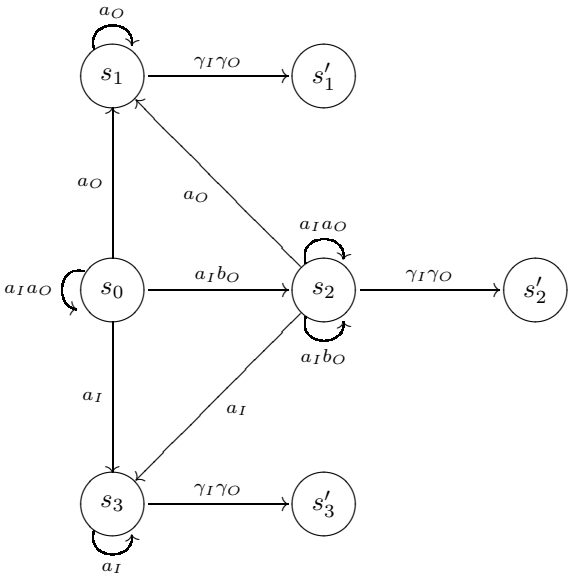

FIGURE 3. Input Output Transition System M

$a = b$. In all cases where we have sequences with length 2, this represents a cycle of length 2. As with M_T, we complete M so that it is input enabled by adding transitions to a new state s'_e from which there are only self-loop transitions whose labels are inputs.

First, note that the set of $\sigma \in L(M)$ that do not contain γ_I includes all those in $(\Sigma_I \cup \Sigma_O \setminus \{\gamma_I, \gamma_O\})^*$ in which input and output alternate, starting with an input, until possibly some suffix that contains either only inputs or only outputs. This is because if we stay in s_0 then the projections on Σ_I and Σ_O are the same while if we leave s_0 then we obtain all such sequences in which the projections on Σ_I and Σ_O differ. Importantly, if we consider the sequences that end in $\gamma_I\gamma_O$ we find that again (before γ_I) input and output alternate, until possibly some suffix that contains either only inputs or only outputs. Further, the sequences of $L(M)$ that end in γ_O are all sequences of the form $\sigma\gamma_I\gamma_O$ for some such σ where σ has different projections on Σ_I and Σ_O.

Now consider the problem of deciding whether $Tr(M_T) \subseteq Tr(M)$ and let us suppose that σ is some shortest sequence in $Tr(M_T) \setminus Tr(M)$. Since M is input-enabled, by the minimality of σ we must have that σ is the label of a path of M_T that does not reach s_e. By Lemma 4.2 we know that every sequence in $L(M_T)$ that does not reach s_e has at least as many inputs as outputs. As a result, all sequences in $L(M_T)$ that do not contain γ_O and are labels of paths that do not reach s_e must also be in $Tr(M)$. Thus, since σ is the label of a path that does not reach s_e, we must have that σ ends in γ_O.

Assume that $\sigma = \sigma'\gamma_I\gamma_O$ for some σ'. If the projections σ'_I and σ'_O on Σ_I and Σ_O respectively (with decorations I and O removed) are different then we can find a sequence σ'' in which input and output alternate until possibly some final suffix that has only input or only output and such that σ' can be formed from σ'' by delaying output. Thus, $\sigma' \in [\sigma''\rangle$. We also know that $\sigma'' \in L(M)$ and so we have that $\sigma' \in Tr(M)$, providing a contradiction. As a result, we must have that the projections of σ' on Σ_I and Σ_O are the same. Observe that $L(M)$ contains no sequences that end in γ_O and have the same projections on Σ_I and Σ_O. In addition, if a sequence in $L(M)$ contains γ_O and reaches state s_e then the input projection of this sequence either does not end in γ_I or contains more than one γ_I. Thus, $Tr(M)$ contains no sequence that ends in $\gamma_I\gamma_O$ and that has the same input and output projections. As a result, $Tr(M_T) \subseteq Tr(M)$ if and only if $L(M_T)$ has no sequence ending in $\gamma_I\gamma_O$ that has the same projections on Σ_I and Σ_O. By Lemma 4.1 this is the case if and only if T does not halt on the empty tape. The result thus follows from the halting problem being undecidable for Turing machines.

Now consider the second part, where we are only concerned with sequences that label paths that end in quiescent states. We can change M slightly to form M' by, for each transition (s, a, s') where s' is not quiescent and $a \notin \{\gamma_I, \gamma_O\}$, adding a transition (s, a, s_e) to the state s_e from which there are only self-loop transitions with input as labels. Then clearly we have that $Tr_\delta(M') = Tr(M)$. Thus, the proof follows in the same way as the first part except using M' rather than M. □

We also have that the corresponding equivalences are undecidable.

THEOREM 4.2. *Let us suppose that M and N are IOTSs with the same input and output alphabets. Then the following are undecidable*

1. $N \equiv M$.
2. $N \equiv_\delta M$.

Proof. Given sets L_1 and L_2 we have that $L_1 \subseteq L_2$ if and only if $L_1 \cup L_2 = L_2$. We can define an IOTS M' such that $L(M') = L(M) \cup L(N)$ and $L_\delta(M') = L_\delta(M) \cup L_\delta(N)$. Thus, $N \sqsubseteq M$ if and only if $M' \equiv M$ and $N \sqsubseteq_\delta M$ if and only if $M' \equiv_\delta M$. The result therefore follows from Theorem 4.1. □

Clearly, these results also show that language inclusion is undecidable for rational semi-commutations even if we restrict attention to those in which the independence relation is of the form $\Sigma_1 \times \Sigma_2$ for sets Σ_1 and Σ_2 that partition the alphabet Σ. It also shows that it is undecidable whether there is a test case that can distinguish an IOTS N, that represents a possible model of the SUT, from the specification IOTS M.

4.2. Decidable cases

We now consider conditions under which inclusion is decidable. We say that a language $L \subseteq Act^*$ is *closed*

under delay if for all $\sigma \in L$ we have that $[\sigma\rangle \subseteq L$. Then the following is clear.

PROPOSITION 4.1. *Given $\sigma \in \Sigma^*$ and language L that is closed under delay we have that $[\sigma\rangle \subseteq L$ if and only if $\sigma \in L$.*

For an IOTS M we have that $Tr(M)$ and $Tr_\delta(M)$ are closed under delay. The following are immediate consequences of the above result.

PROPOSITION 4.2. *Given IOTSs N and M with the same input and output alphabets we have that $Tr(N) \subseteq Tr(M)$ if and only if $L(N) \subseteq Tr(M)$. Further, $Tr_\delta(N) \subseteq Tr_\delta(M)$ if and only if $L_\delta(N) \subseteq Tr_\delta(M)$.*

Thus, if $Tr(M)$ is regular and we can construct a corresponding finite automaton then we can decide whether $Tr(N) \subseteq Tr(M)$. Given a dependence relation D and a sequence $\rho \in \Sigma^*$, we can define the corresponding directed graph $G(D, \rho)$ in which the vertices represent the elements of Σ that are in ρ and there is an edge from the vertex representing a (that is in ρ) to the vertex representing b (that is in ρ) if and only if $(a, b) \in D$.

It has been proved that given a semi-commutation with dependence relation D and finite automaton M, $Tr(M)$ is regular if every cycle of M has a label ρ such that the directed graph $G(D, \rho)$ is strongly connected[4] [32]. In addition, the proof of this results shows how the finite automaton M' with $L(M') = Tr(M)$ can be constructed.

It is straightforward to see that since we have that $D = (I \times O) \cup (I \times I) \cup (O \times O)$, $G(D, \rho)$ is strongly connected if and only if either ρ only contains inputs or ρ only contains outputs.

This, combined with Proposition 4.2, leads to the following result.

PROPOSITION 4.3. *Let us suppose that we have IOTS M and N and every cycle of M has a label ρ such that either ρ only contains inputs or ρ only contains outputs. Then it is decidable whether $Tr(N) \subseteq Tr(M)$.*

Naturally, the corresponding result also holds for deciding whether $Tr_\delta(N) \subseteq Tr_\delta(M)$ since we can again construct a finite automaton M' with $L(M') = Tr_\delta(M)$ if every cycle of M has a label ρ such that $G(D, \rho)$ is strongly connected.

Note, however, that while this gives a condition under which conformance is decidable (and thus, so is equivalence), the construction of the finite automaton M' need not operate in polynomial time. We now consider an alternative condition under which the problem can be solved in polynomial time.

Sometimes we consider AIOTSs rather than IOTSs. This introduced two restrictions. First, input and output alternate, with an input leading to an output response. Second, the AIOTS cannot receive further input between receiving x and sending y. For AIOTSs we have the following result.

PROPOSITION 4.4. *Given AIOTSs M and N with the same input and output alphabets, $Tr(N) \subseteq Tr(M)$ if and only if $L(N) \subseteq L(M)$.*

Proof. This follows from observing that every σ in $L(N)$ or $L(M)$ is of the form $?i_1!o_1 \ldots ?i_k!o_k \in (IO)^*$ or the form $?i_1!o_1 \ldots !o_{k-1}?i_k \in (IO)^*I$ and that σ can be reconstructed from any element of $[\sigma\rangle$: if $\sigma' \in [\sigma_1\rangle \cap [\sigma_2\rangle$ for $\sigma_1, \sigma_2 \in L(N) \cup L(M)$ then $\sigma_1 = \sigma_2$. □

We therefore obtain the following.

PROPOSITION 4.5. *Let us suppose that M and N are AIOTSs with the same input and output alphabets and that M has m states and N has n states. The problem of determining whether $Tr(N) \subseteq Tr(M)$ can be solved in $O(nm)$ time.*

Proof. By Proposition 4.4 it is sufficient to decide whether $L(N) \subseteq L(M)$. In order to do this it is sufficient to produce the produce machine in which each state is of the form (s, s'), where s is a state of N and s' is a state of M. The transitions of the product machine are defined by synchronising on common actions except where an output in N is not allowed from the corresponding state of M and then we move to a special state s_f. Clearly, $L(N) \subseteq L(M)$ if and only if s_f is not reachable and since the product machine has $O(mn)$ states we can decide this in $O(nm)$ by using a depth-first search [38]. □

4.3. Summary

This section has shown that implementation relations for asynchronous communications are undecidable. There are several consequences of these results for asynchronous testing. First, the results show that it is undecidable whether there is a test case that can distinguish between the specification IOTS M and an IOTS N that represents a possible behaviour of the SUT. This means that we cannot expect test generation algorithms based on fault domains to extend to asynchronous testing. In addition, some test generation algorithms for state-based systems use test cases that distinguish states of the specification in order, for example, to check that the state after a sequence in Act^* is correct. Again, the result suggests that these types of test generation algorithms cannot be extended to asynchronous testing. However, we also gave conditions under which the implementation relations are decidable and it may well be possible to extend test generation algorithms to asynchronous testing under such conditions.

[4] A directed graph is strongly connected if for any two vertices v and v' it is possible to find a path from v to v'.

5. CHECKING INTERSECTION

We have seen that conformance is undecidable for implementation relations $\sqsubseteq$ and $\sqsubseteq_\delta$ but have also given conditions under which it is decidable. Sometimes we are interested in knowing whether any behaviour of a model M satisfies a given property P and this is the problem we consider in this section. In this section we therefore consider the situation in which we have two models: an IOTS M representing the system and a FA P that represents a property of interest. In this section we consider two scenarios when communications are asynchronous. In the first, given IOTS M and FA P we want to know whether $Tr(M) \cap Tr(P) = \emptyset$. This asks whether any observation that might be made of P, through asynchronous FIFO channels, might also be made of M. We then consider the problem of deciding whether $Tr(M) \cap \text{Ł}(P) = \emptyset$ and thus we are asking whether any sequence in the regular language defined by P is a possible observation of M when communicating with M through asynchronous FIFO channels.

In the first case we are asking whether there are observations that can be made of M that are consistent with P, a problem that might be of interest if P models some undesirable behaviours and we wish to know whether the observations regarding M can be consistent with such behaviours. In the second case, the sequences in $L(P)$ might correspond to sequences that are undesirable since, for example, M will interact through FIFO channels with another system M' and M' should not receive such sequences.

Since M is a model of a system, given as an IOTS, $L(M)$ is prefix closed. Thus, if we see M as a finite automaton then all states of M are final states. However, P will represent a set of behaviours in which we are interested and thus $L(P)$ need not be prefix closed. We therefore restrict attention to the case where M is an IOTS and P is a FA. Naturally, we can also consider the language $Tr_\delta(M)$.

We use Post's Correspondence Problem to prove that both problems are undecidable.

DEFINITION 5.1. *Given sequences $\alpha^1, \ldots, \alpha^m$ and $\beta^1, \ldots, \beta^m$ over an alphabet Σ, Post's Correspondence Problem (PCP) is to decide whether there is a sequence $i_1, \ldots, i_k$ of indices from $[1, m]$ such that $\alpha^{i_1} \ldots \alpha^{i_k} = \beta^{i_1} \ldots \beta^{i_k}$.*

The following is known [39].

THEOREM 5.1. *Post's Correspondence Problem is undecidable.*

We now prove that the first type of problem described above is undecidable.

THEOREM 5.2. *Given IOTS M and FA P the following problems are undecidable.*

1. $Tr(M) \cap Tr(P) = \emptyset$
2. $Tr_\delta(M) \cap Tr(P) = \emptyset$

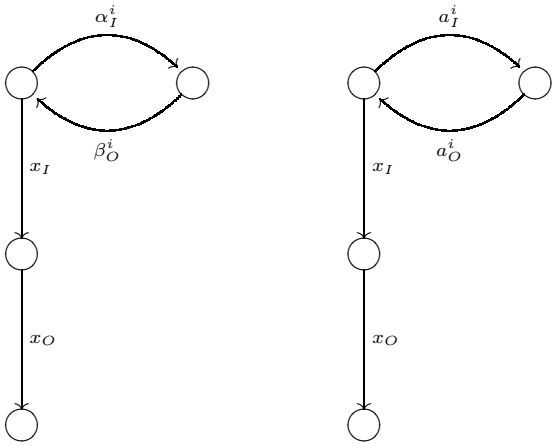

FIGURE 4. IOTS M and FA P

Proof. We start with the first result. We assume that we have an instance of PCP given by sequences $\alpha^1, \ldots, \alpha^m$ and $\beta^1, \ldots, \beta^m$ over alphabet $\Sigma = \{a^1, \ldots, a^k\}$ and show how we can represent this in terms of checking whether $Tr(M) \cap Tr(P) = \emptyset$ for an IOTS M and finite automaton P.

We define copies Σ_I and Σ_O of Σ and use Σ_I for inputs and Σ_O for outputs. As before, given $\sigma \in \Sigma^*$ we let σ_I denote the sequence formed by replacing elements of Σ by corresponding elements of Σ_I and we let σ_O denote the sequence formed by replacing elements of Σ by corresponding elements of Σ_O.

M and P have alphabet $\Sigma_I \cup \Sigma_O \cup \{x_I, x_O\}$ (some $x \notin \Sigma$), where x_I is an input and x_O is an output. We define M such that $L(M)$ is the set of sequences defined by the regular expression $(\alpha_I^1 \beta_O^1 + \ldots + \alpha_I^m \beta_O^m)^* x_I x_O$ and all prefixes of such sequences. As before, we assume M has been completed through the addition of state s_e and corresponding transitions. Further, we let P be such that $L(P)$ is the set of sequences defined by the regular expression $(a_I^1 a_O^1 + \ldots + a_I^k a_O^k)^+ x_I x_O$. The FA M and P are illustrated in Figure 4.

Now consider the problem of deciding whether $Tr(M) \cap Tr(P)$ is empty. First note that all elements of $Tr(P)$ contain at least one element of Σ_I and at least one element of Σ_O, in addition to $x_I x_O$, and so $x_I x_O$ is not in $Tr(M) \cap Tr(P)$. Thus, $Tr(M) \cap Tr(P)$ is non-empty if and only if there is a sequence in $(\alpha_I^1 \beta_O^1 + \ldots + \alpha_I^m \beta_O^m)^+ x_I x_O$ that has projections on Σ_I and Σ_O that correspond to the same sequence in Σ^* and this is the case if and only if there is a solution to this instance of the PCP. The result thus follows from Theorem 5.1.

The second result follows similarly since all sequences of $L(M)$ that end in x_O are labels of quiescent paths. □

As we have seen, if we cannot know when all inputs and outputs of a sequence have been observed then we

might instead reason about the language $Tr^p(M)$ for IOTS M. The following shows, however, that there is no value in asking whether $Tr^p(M) \cap Tr^p(P) \neq \emptyset$.

PROPOSITION 5.1. *Given IOTS M and FA P, if $L(P)$ is non-empty then $Tr^p(M) \cap Tr^p(P) \neq \emptyset$.*

Proof. Assume that $L(P)$ is non-empty. First, consider the case where there is a sequence in $L(P)$ that contain one or more inputs. Then we must have a sequence in $Tr^p(P)$ that contain only inputs and this must also be in $Tr^p(M)$ since M is input-enabled. Thus, if there is a sequence in $L(P)$ that contains an input then we must have that $Tr^p(M) \cap Tr^p(P) \neq \emptyset$. In addition, if no sequence in $L(P)$ contains inputs then since $L(P)$ is non-empty, $L(P)$ must contain sequences from O^*. As a result, $Tr^p(P)$ must contain the empty sequence and so, since $L(M)$ is prefix closed, $Tr^p(M) \cap Tr(P) \neq \emptyset$. □

We now prove that the second type of problem described above is undecidable for the languages $Tr(M)$, $Tr^p(M)$, and $Tr_\delta(M)$.

THEOREM 5.3. *Given IOTS M and FA P the following problems are undecidable.*

1. $Tr(M) \cap L(P) = \emptyset$
2. $Tr^p(M) \cap L(P) = \emptyset$
3. $Tr_\delta(M) \cap L(P) = \emptyset$

Proof. We start with the first problem. We assume that a Turing Machine T has been given and use the IOTS $\bar{M}_T$ which can be formed from the IOTS M_T (defined in Section 3) by switching input and output decorations. Note that in forming $\bar{M}_T$ we do not include copies of the transitions from M_T that were added to make M_T input-enabled since these would add output, instead we add a state s_e and transitions to make $\bar{M}_T$ input enabled. Thus, from Lemma 4.1 we know that $L(\bar{M}_T)$ contains a sequence ρ that ends in $\gamma_O\gamma_I$ and has the same projections on Σ_I and Σ_O if and only if these projections define a halting computation.

Let $\Sigma = \{a^1, \ldots, a^k\}$ and define P such that $L(P) = (a^1_O a^1_I + \ldots a^k_O a^k_I)^* \gamma_O \gamma_I$. We know, from Lemma 4.2, that all sequences in $L(\bar{M}_T)$ that end in $\gamma_O\gamma_I$ contain at least as many outputs as inputs. Thus, there is a sequence $\rho \in L(\bar{M}_T)$ that ends in $\gamma_O\gamma_I$ and has the same projections on Σ_I and Σ_O if and only if $Tr(\bar{M}_T)$ contains such a sequence that starts with output and in which output and input alternate, since we can form such a sequence by delaying output. Thus, $Tr(\bar{M}_T) \cap L(P) \neq \emptyset$ if and only if there is a sequence $\rho \in L(\bar{M}_T)$ that ends in $\gamma_O\gamma_I$ and has the same projections on Σ_I and Σ_O. By Lemma 4.1 this is the case if and only if T has a halting computation and so the result follows.

For the second result we slightly change $\bar{M}_T$ by swapping the labels of the transitions in the path with label $\gamma_O\gamma_I$ to form $\bar{M}'_T$. We also set define P' so that $L(P') = (a^1_O a^1_I + \ldots a^k_O a^k_I)^* \gamma_I \gamma_O$. Observe that all elements of $L(P')$ end in $\gamma_I\gamma_O$ and contain γ_O and γ_I exactly once each. For a sequence $\sigma \in Tr^p(\bar{M}'_T)$ to be of this form there must be some $\sigma' \in L(\bar{M}'_T)$ such that $\sigma' \rightsquigarrow^p \sigma$ and σ' ends in $\gamma_I\gamma_O$. In addition, since σ' and σ both end in $\gamma_I\gamma_O$ and contain γ_I and γ_O exactly once, we must have that $\sigma' \rightsquigarrow \sigma$. Thus, $Tr^p(\bar{M}'_T) \cap L(P) \neq \emptyset$ if and only if there is a sequence $\rho \in L(\bar{M}'_T)$ that ends in $\gamma_O\gamma_I$ and has the same projections on Σ_I and Σ_O and so the result follows.

For the third result it is sufficient to observe that sequences of $\bar{M}_T$ that end in $\gamma_O\gamma_I$ reach a quiescent state. □

These problems are decidable under the conditions considered previously. The proof of the following is similar to that of Proposition 5.2: the conditions given ensure that we can construct a finite automaton M' such that $Tr(M) = L(M')$.

PROPOSITION 5.2. *Let us suppose that we have IOTS M and finite automaton P and every cycle of M has a label ρ such that either ρ only contains inputs or ρ only contains outputs. Then it is decidable whether $Tr(M) \cap L(P) = \emptyset$. In addition, if every cycle of P has a label ρ such that either ρ only contains inputs or ρ only contains outputs, then it is also decidable whether $Tr(M) \cap Tr(P) = \emptyset$.*

The proof of the following is equivalent to that of Proposition 4.4.

PROPOSITION 5.3. *Given AIOTS M and alternating FA P with the same input and output alphabets, $Tr(M) \cap Tr(P) = \emptyset$ if and only if $L(M) \cap L(P) = \emptyset$.*

We therefore obtain the following.

PROPOSITION 5.4. *Let us suppose that M is an AIOTS and P is an alternating FA with the same input and output alphabets and let m be the number of states of M and n the number of states of N. The problem of determining whether $Tr(M) \cap Tr(P) = \emptyset$ can be solved in $O(mn)$ time.*

Proof. By Proposition 5.3 it is sufficient to decide whether $L(M) \cap L(P) = \emptyset$. But, this can be achieved by forming the product automaton from M and P and deciding whether it is possible to reach a state (s, s') such that s' is a final state of P. □

6. CONCLUSIONS

This paper has explored problems in the context of analysing input output transition system (IOTS) models for systems that interact with their environment through asynchronous FIFO channels. The motivation was testing and either we want to know whether one model conforms to another or whether the languages defined by two models intersect. While it is possible to compose an IOTS with models of the channels, even with bounded channels this can lead to a state space explosion. We therefore investigated alternative approaches based on defining a language

that corresponds to the set of observations that might be made regarding an IOTS. Given IOTS M this language, $Tr(M)$, is the set of sequences that can be formed from sequences of M by delaying output.

Previous work has defined implementation relations for testing through FIFO channels but these have required that quiescence is observable. However, in order to observe quiescence it is necessary to place bounds on message latency and this appears not to fit well with asynchronous communications. We therefore did not assume that quiescence can be observed and defined implementation relations, $\sqsubseteq$, $\sqsubseteq^p$, and $\sqsubseteq_\delta$. Under $\sqsubseteq$ we require that all sequences that might be observed when interacting with N through FIFO channels are also sequences that might be observed when interacting with M through FIFO channels ($Tr(N) \subseteq Tr(M)$). The implementation relation $\sqsubseteq^p$ weakened this by allowing for the possibility that an observation is partial: input sent to the SUT might not have been received yet and output produced by the SUT might not have been observed yet. Finally, $\sqsubseteq_\delta$ is similar to $\sqsubseteq$ except that it only considers sequences that can lead to a quiescent state: one from which the IOTS cannot change state or produce output without receiving further input.

It transpired that for any IOTSs N and M we have that $N \sqsubseteq M$ if and only if $N \sqsubseteq^p M$. We proved that given IOTSs N and M with finite sets of states and actions, for each implementation relation it is undecidable whether N conforms to M and whether N is equivalent to M. In proving this, we also proved that inclusion and equivalence are undecidable for rational semi-commutations whose independence relation is of the form $\Sigma_1 \times \Sigma_2$ for Σ_1 and Σ_2 that partition the alphabet Σ. This problem appears not to have been previously considered, possibly because semi-commutations are a generalisation of partial commutations and inclusion is known to be undecidable for partial commutations. However, the class of semi-commutation we considered requires the independence relation to be anti-symmetric while in a partial commutation it is symmetric and so we do not directly inherit the result from partial commutations. We also proved that conformance and equivalence are decidable in low order polynomial time for alternating models, a type of model and system that has been widely studied [24, 25, 26].

One consequence of the results is that given two IOTSs M and N, where M might represent the specification and N a (possible faulty) implementation, it is undecidable whether there is a test case that can distinguish N from M. This also has ramifications for test generation based on a fault domain $\mathcal{F}$. Such test generation techniques, for synchronous communications, return a test suite that distinguishes between the specification M and all faulty elements of $\mathcal{F}$ and so have a form of guaranteed fault detection power. The results in this paper indicate that we cannot expect to be able to extend such test generation methods to asynchronous communications in general, but this may be possible for some classes of IOTS.

We also considered the situation in which as well as an IOTS model M of the system we have a property represented by a finite automaton P and we want to know whether any element of $Tr(M)$ is consistent with P. We defined two versions of this problem. In one, we ask whether $Tr(M) \cap Tr(P)$ is non-empty for IOTS M and finite automaton P that represents the property of interest. In the second, we ask whether $Tr(M) \cap L(P)$ is non-empty. Similar problems have been considered for message sequence charts [28, 29]. Both types of problem are undecidable but, again, can be decided in low-order polynomial time for alternating models.

There are several lines of future work. First, we assumed that the communications channels are FIFO and it would be interesting to investigate alternative assumptions. Second, while we have given conditions under which the problems considered are decidable, there should be scope to weaken these. We have seen that conformance is decidable for alternating models and it would be interesting to extend this to conditions under which the use of asynchronous channels does not affect our implementation relations: such conditions might be seen as testability conditions for systems that interact with their environment through FIFO channels. Potentially, this should relate to recent work that investigated conditions under which a test case for synchronous testing can be used in asynchronous testing [40, 41]. Finally, there are conceptual similarities between the issues addressed in this paper and partial order reduction and it would be interesting to explore these further.

ACKNOWLEDGEMENTS

I would like to thank the reviewers for their constructive comments that strengthened the paper. In particular, I would like to that the anonymous reviewer who pointed out that message latency is not the only reason for having to consider the asynchronous case.

REFERENCES

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

[2] v. Bochmann, G., Das, A., Dssouli, R., Dubuc, M., Ghedamsi, A., and Luo, G. (1992) Fault models in testing. *Protocol Test Systems IV*, pp. 17–30. Elsevier Science (North-Holland).

[3] Fujiwara, S. and v. Bochmann, G. (1991) Testing non-deterministic state machines with fault coverage. *Proceedings of Protocol Test Systems, IV*, pp. 267–280.

[4] Gonenc, G. (1970) A method for the design of fault detection experiments. *IEEE Transactions on Computers*, **19**, 551–558.

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

[6] Hierons, R. M. (2002) Comparing tests in the presence of hypotheses. *ACM Transactions on Software Engineering and Methodology*, **11**, 427–448.

[7] Hierons, R. M. and Ural, H. (2006) Optimizing the length of checking sequences. *IEEE Transactions on Computers*, **55**, 618–629.

[8] Hierons, R. M. (2010) Checking experiments for stream X-machines. *Theoretical Computer Science*, **411**, 3372–3385.

[9] Ipate, F. and Holcombe, M. (1997) An integration testing method that is proved to find all faults. *International Journal of Computer Mathematics*, **63**, 159–178.

[10] Kuhn, D. R. (1999) Fault classes and error detection capability of specification-based testing. *ACM Transactions on Software Engineering and Methodology*, **8**, 411–424.

[11] Moore, E. F. (1956) Gedanken-experiments. In Shannon, C. and McCarthy, J. (eds.), *Automata Studies*. Princeton University Press.

[12] Tretmans, J. (1996) Conformance testing with labelled transitions systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, **29**, 49–79.

[13] Tretmans, J. (2008) Model based testing with labelled transition systems. *Formal Methods and Testing*, Lecture Notes in Computer Science, **4949**, pp. 1–38. Springer.

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

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

[16] Hierons, R. M. Oracles for distributed testing. *IEEE Transactions on Software Engineering*, **to appear**.

[17] Jard, C., Jéron, T., Tanguy, L., and Viho, C. (1999) Remote testing can be as powerful as local testing. *Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, IFIP Conference Proceedings, **156**, pp. 25–40. Kluwer.

[18] Tretmans, J. (1992) A formal approach to conformance testing. PhD thesis University of Twente, Netherlands.

[19] Tretmans, J. (1996) Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, **17**, 103–120.

[20] da Silva Simão, A. and Petrenko, A. (2011) Generating asynchronous test cases from test purposes. *Information and Software Technology*, **53**, 1252–1262.

[21] Bhateja, P., Gastin, P., and Mukund, M. (2006) A fresh look at testing for asynchronous communication. *Automated Technology for Verification and Analysis, 4th International Symposium (ATVA 2006)*, Lecture Notes in Computer Science, **4218**, pp. 369–383. Springer.

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

[23] Huo, J. and Petrenko, A. (2009) Transition covering tests for systems with queues. *Software Testing, Verification and Reliability*, **19**, 55–83.

[24] Alur, R., Henzinger, T. A., and Kupferman, O. (1997) Alternating-time temporal logic. *38th Annual Symposium on Foundations of Computer Science (FOCS 1997)*, pp. 100–109. IEEE Computer Society.

[25] Alur, R., Henzinger, T. A., and Kupferman, O. (2002) Alternating-time temporal logic. *Journal of the ACM*, **49**, 672–713.

[26] Henzinger, T. A. and Prabhu, V. S. (2006) Timed alternating-time temporal logic. *4th International on Conference Formal Modeling and Analysis of Timed Systems (FORMATS 2006)*, Lecture Notes in Computer Science, **4202**, pp. 1–17. Springer.

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

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

[29] Genest, B., Muscholl, A., Seidl, H., and Zeitoun, M. (2006) Infinite-state high-level MSCs: Model-checking and realizability. *Journal of Computer and Systems Science*, **72**, 617–647.

[30] Diekert, V. and Mtivier, Y. (1997) Partial commutation and traces. *Handbook of formal languages, vol. 3: beyond words*, pp. 457–533.

[31] Mazurkiewicz, A. W. (1984) Traces, histories, graphs: Instances of a process monoid. In Chytil, M. and Koubek, V. (eds.), *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, **176**, pp. 115–133. Springer.

[32] Clerbout, M. and Latteux, M. (1987) Semi-commutations. *Information and Computation*, **73**, 59–74.

[33] Petrenko, A., Yevtushenko, N., and Huo, J. (2003) Testing transition systems with input and output testers. *15th IFIP International Conference on Testing of Communicating Systems (TestCom 2003)*, Lecture Notes in Computer Science, **2644**, pp. 129–145. Springer.

[34] Clerbout, M. and Gonzalez, D. (1994) Atomic semicommutations. *Theoretical Computer Science*, **123**, 259–272.

[35] Clerbout, M., Latteux, M., and Roos, Y. (1995) Semi-commutations. *The Book of Traces*, pp. 487–552.

[36] Floyd, R. W. (1964). New proofs and old theorems in logic and formal linguistics. Computer Associated Inc, Wakefield, Mas.

[37] Davis, M. D. and Weyuker, E. J. (1983) *Computability, Complexity and Languages*. Academic Press.

[38] Tarjan, R. E. (1972) Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, **1**, 146–160.

[39] Post, E. L. (1946) A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, **52**, 264–268.

[40] Noroozi, N., Khosravi, R., Mousavi, M. R., and Willemse, T. A. C. (2011) Synchronizing asynchronous conformance testing. *9th International Conference on Software Engineering and Formal Methods (SEFM 2011)*, Lecture Notes in Computer Science, **7041**, pp. 334–349. Springer.

[41] Weiglhofer, M. and Wotawa, F. (2009) Asynchronous input-output conformance testing. *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009)*, pp. 154–159. IEEE Computer Society.