An extended framework for passive asynchronous testing[☆]

Robert M. Hierons

Department of Computer Science, Brunel University London, United Kingdom

Mercedes G. Merayo

Dep. de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain

Manuel Núñez

Dep. de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain

Abstract

In passive testing a monitor observes the trace (sequence of inputs and outputs) of the system under test (SUT) and checks that this trace satisfies a given property P, potentially triggering a response if an incorrect behaviour is observed. Recent work has explored a variant of passive testing, in which we have a required property P of the traces of the SUT and there is a first-in-first-out (FIFO) network between the SUT and the monitor. The problem here is that the trace observed by the monitor need not be that produced by the SUT. Previous work has shown how such asynchronous passive testing can be performed if the property P is defined by a pair (ρ, O_ρ) that represents the requirement that if trace ρ is produced by the SUT then the next output must be from the set O_ρ. This paper generalises the previous work to the case where the property P is defined by a finite automaton.

Keywords: Formal approaches to testing; passive testing; asynchronous testing.

1. Introduction

Software testing [1, 26] is one of the most important forms of verification and validation and plays a fundamental role in the production of quality software. Normally, testing involves executing the system under test (SUT) with a set

[☆]Research partially supported by the Spanish projects ESTuDIo (TIN2012-36812-C02-01) and DArDOS (TIN2015-65845-C3-1), the Comunidad de Madrid project SICOMORo-CM (S2013/ICE-3006) and the UCM program to fund research groups (910606).

of test cases and then checking that the observed behaviour are acceptable. Test cases must be carefully selected from the huge, and potentially infinite, set of available test cases. However, sometimes we cannot apply such an active approach. This is the case, for example, if the SUT is 'live' and the application of test cases is not allowed since it might corrupt the system state or take resources away from users. In such situations, one might instead apply *passive testing*, which is a process in which the behaviour of the SUT is observed by a monitor and the monitor checks that the observations satisfy certain required properties. Typically, these properties are relatively simple and do not form a complete specification: this allows the monitor to apply checks in real-time and for the implementation of the monitor to consume relatively few resources. It is worth mentioning that both active and passive testing can be given a mathematical basis and that formal approaches to testing is an active research area [9, 14].

Two related terms have been used in the literature: *passive testing* [2, 3, 7, 10, 21, 22, 25, 27] and *runtime monitoring* [4, 5, 6, 12, 18, 19, 20, 23]. These terms are used by different communities and both have associated literature going back well over 10 years. The main difference between these lines of work is the way in which the property of interest is expressed: in runtime monitoring the property is usually a linear temporal logic (LTL)[1] formula and in passive testing the property is usually describes by a finite automaton possibly with data added.

In runtime monitoring (and also passive testing) one observes finite traces. Thus, it does not make sense to check properties, such as liveness properties, where counter-examples are defined in terms of infinite traces of the SUT. As a result, the focus of the runtime monitoring community has been on *safety properties*, where every infinite trace that fails a required property has some finite prefix σ such that all extensions of σ fail the property (see, for example, [18]). The trace σ thus provides a finite witness to the failure. While runtime monitoring uses LTL rather than finite automata, and a property is expressed in terms of a Büchi automaton, it is known that one can use a finite automaton to recognise the 'bad' finite prefixes of a safety property [18, 19]. While the generation of a finite automaton from a safety property can take double exponential time, this becomes single exponential if the properties are not pathological [18, 19]. In addition, it has been found that in practice it is feasible to generate a finite automaton of reasonable size from a non-pathological safety property [20]. As a result, we can formulate most approaches to passive testing and runtime monitoring in terms of checking a property defined by a finite automaton M and so this is the scenario we consider. Note that in situations in which passive testing is applied, we typically do not know when system execution started and so properties will normally be of the form $\Box P$ (stating that P is globally true).

In this paper we investigate the situation in which we have a safety property

[1]The focus is on *linear* temporal logics since the monitor simply observes a trace and so there is not scope for branching.

$\Box P$, a finite automaton M that represents the set of finite traces that violate P, and there is an asynchronous first-in-first-out (FIFO) channel between the SUT and the monitor. As a result of the channel being asynchronous, the monitor observes inputs before they are received by the SUT and observes outputs after they were produced by the SUT. Thus, the trace produced by the SUT need not be the one observed by the monitor: outputs may be delayed. While there has been much work on both passive testing and runtime monitoring, almost all of this work has considered the synchronous case where we directly observe the actions in which the SUT participates and so does not make a distinction between input and output.

This paper generalises our previous work [16] in which we only considered properties defined by a pair (ρ, O_ρ) that states that if the trace ρ is produced by the SUT then the next output produced by the SUT must be from O_ρ. This previous approach showed how one can generate a property, defined by a finite automaton, that the trace observed by the monitor must satisfy. In Section 3 we describe an alternative approach in which we take the trace σ observed by the monitor and define a finite automaton $\mathcal{M}(\sigma)$ that accepts the set of traces that the SUT might have produced: those that could have resulted in the observation of σ. Given finite automaton M that defines the property of interest, the problem reduces to deciding whether the intersection of the languages defined by $\mathcal{M}(\sigma)$ and M is empty. This provides a general solution to the problem but has the disadvantage that the size of $\mathcal{M}(\sigma)$ depends on the length of the trace σ. In contrast, our previous approach rewrote the property of interest and so the size of the resultant finite automaton depended on the size of the property.

Since one would expect a property used to be relatively small, it is desirable to have approaches that operate on the finite automaton M defining the property rather than on the observed trace σ. The rest of the paper thus focuses on generalising our previous approach, in which a property was defined by a pair (ρ, O_ρ). It has previously been proved that if M is a finite automaton then the language $\mathcal{L}(M)$, of observations that might result from traces of M, need not be regular [16]. As a result, there is no general approach that operates on the finite automaton M defining the property of interest and so we consider conditions under which $\mathcal{L}(M)$ is regular. In Section 4 we first generalise previous work to the case where M is acyclic. This is extended in Section 5 to the case where every cycle in M has either only inputs or only outputs (but there can be cycles with inputs and also cycles with outputs). Throughout this paper we will use the term *passive testing* to refer to the situation in which we have a finite automaton M that defines the set of finite traces that violate the property of interest but, as noted above, this also captures a number of scenarios in runtime monitoring.

The paper is structured as follows. In Section 2 we define terminology and notation used throughout the paper. Section 3 describes the general approach, which operates on the trace σ. In Section 4 we explore the situation in which M is acyclic. Section 5 then generalises this to the case where every cycle in M has either only inputs or only outputs. Finally, Section 6 draws conclusions

Figure 1: Architecture

and discusses possible lines of future work.

2. Preliminaries: systems and observations

In passive testing a monitor observes actions in which the SUT participates. When interaction is synchronous and the monitor observes the actual sequence of actions produced by the SUT then there is no need to distinguish between input and output. However, there might be a network between the monitor and the SUT (Figure 1) with the communications being asynchronous. There is then some asymmetry: inputs are observed by the monitor before they are received by the SUT and outputs are observed by the monitor after they are produced by the SUT. As a result, we need to distinguish between inputs and outputs. Throughout this paper we let I denote the set of inputs that the SUT might received, O denote the set of outputs that it might produce, and we let $\mathcal{A}ct = I \cup O$ be the set of actions. We will assume that I and O are disjoint and finite. In addition, we will usually precede the name of an input by ? and the name of an output by !.

We will use finite automata to represent properties that the monitor checks.

Definition 1. *A* finite automaton (FA) $M = (Q, \mathcal{A}ct, T, q_{in}, F)$ *is a tuple in which* Q *is a finite set of states,* $q_{in} \in Q$ *is the initial state,* $\mathcal{A}ct$ *is a finite set of actions,* $T \subseteq Q \times (\mathcal{A}ct \cup \{\epsilon\}) \times Q$ *is the transition relation, and* $F \subseteq Q$ *is the set of final states. A transition* $(q, a, q') \in T$ *for* $a \in \mathcal{A}ct$ *means that from state* q *it is possible to move to state* q' *with action* $a \in \mathcal{A}ct$*. A transition* $(q, \epsilon, q') \in T$ *means that from state* q *it is possible to move to state* q' *with no action (a hidden transition). We use the following notation concerning the performance of (sequences of) actions.*

- *A sequence* $\rho = (q_1, a_1, q_2)(q_2, a_2, q_3) \dots (q_k, a_k, q_{k+1})$ *of consecutive transitions is said to be a* walk *of* M *and the* label *of this walk is* $a_1 \dots a_k$ *with all instances of* ϵ *removed. If* $q_1 = q_{k+1}$ *then* ρ *is a* cycle*. If all of the* q_i *are distinct then* ρ *is a* path*.*
- *If* $(q, a, q') \in T$*, for* $a \in \mathcal{A}ct \cup \{\epsilon\}$*, then we write* $q \xrightarrow{a} q'$ *and* $q \xrightarrow{a}$*.*
- *Let us suppose that there exist* $q_0, \dots, q_m$*,* $q = q_0$*,* $q' = q_m$ *such that for all* $0 \leq i < m$ *we have that* $q_i \xrightarrow{a_{i+1}} q_{i+1}$*. Then we write* $q \stackrel{\sigma}{\Longrightarrow} q'$ *for the trace* $\sigma \in \mathcal{A}ct^*$ *formed by removing all instances of* ϵ *from* $a_1 \dots a_m$*. Note that* $q \stackrel{\epsilon}{\Longrightarrow} q$*, where we overload* ϵ *to also denote an empty sequence.*

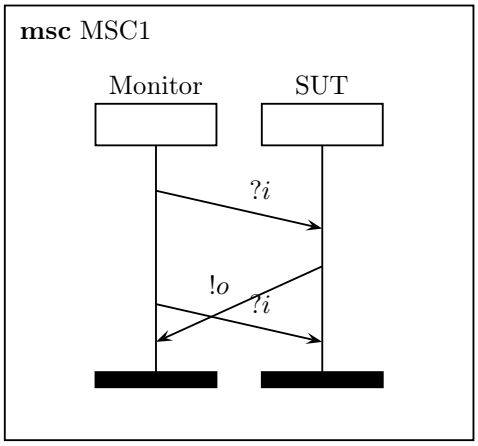


Figure 2: Observation of $?i?i!o$ despite the SUT producing $?i!o?i$

- *If there exists* $q' \in Q$ *such that* $q_{in} \stackrel{\sigma}{\Longrightarrow} q'$ *then we write* $M \stackrel{\sigma}{\Longrightarrow}$.
- *If there exists* $q' \in F$ *such that* $q_{in} \stackrel{\sigma}{\Longrightarrow} q'$ *then we say that* σ *is a* trace *of* M. *We let* $L(M)$ *denote the set of traces of* M.

We will assume that a finite automaton that defines a property does not have transitions labelled with ϵ; these are included in the above definition since we will construct FA with ϵ transitions. Note that it is possible to eliminate ϵ transitions in polynomial time [17].

We assume a framework where communications between entities are asynchronous and first-in-first-out (FIFO). If the SUT produces a trace σ then the actual trace σ' observed might be one formed from σ by delaying outputs. Thus, if a system performs a certain trace then we can observe a variation of this trace where the outputs appear later than they were actually performed. This is due to the fact that the monitor observes inputs before they are received by the SUT and observes outputs after they are produced by the SUT. For example, if the monitor observes the trace $?i?i!o$, in which $?i$ is an input and $!o$ is an output, then it is possible that the SUT actually produced either $?i!o?i$ or $!o?i?i$ and that the observation of $?i?i!o$ was due to the delaying of output. The first of these scenarios is illustrated in Figure 2 in which vertical lines represent processes, arcs represent messages, and time progresses as we move down a vertical line. The previous explanation leads to the following definition of the traces that can be observed by a monitor if the SUT produces σ and also the corresponding language of possible observations regarding the SUT [16].

Definition 2. *Let* $\sigma, \sigma' \in \mathcal{A}ct^*$ *be sequences of actions. We say that* σ' *is an observation of* σ, *denoted by* $\sigma \rightsquigarrow \sigma'$, *if there exist sequences* $\sigma_1, \sigma_2 \in \mathcal{A}ct^*$, $!o \in O$ *and* $?i \in I$ *such that* $\sigma = \sigma_1 !o?i\sigma_2$ *and* $\sigma' = \sigma_1 ?i!o\sigma_2$. *We let* $\mathcal{L}(\sigma)$ *denote the set of traces that can be formed from* σ *through sequences of transformations*

of the form $\rightsquigarrow$*, that is,* $\mathcal{L}(\sigma) = \{\sigma' | \sigma \rightsquigarrow^* \sigma'\}$*. We will overload this to say that given an FA* M*,* $\mathcal{L}(M) = \cup_{\sigma \in L(M)} \mathcal{L}(\sigma)$ *is the set of traces that might be observed when interacting with* M *through asynchronous FIFO channels.*

Given trace σ we therefore have that $\mathcal{L}(\sigma)$ is the set of traces that might be observed by the monitor if the SUT produces σ. Further, if M recognises a set of traces that should not occur then $\mathcal{L}(M)$ is the set of traces that might be observed by the monitor if the SUT produces a trace that violates the property defined by M.

In our previous work [16] we considered properties of the form (ρ, O_ρ) that state that if the trace ρ is produced by the SUT then the next output produced by the SUT must be from O_ρ. We showed how, given ρ[2], we can construct a finite automaton that accepts all elements of $\mathcal{L}(\rho)$ and also any trace that could have resulted from the SUT producing a trace that has ρ as a sub-trace (i.e. all elements in $\cup_{\rho_1, \rho_2 \in \mathcal{A}ct^*} \mathcal{L}(\rho_1 \rho \rho_2)$). This paper generalises our first approach to consider the case where a property of interest (defining a set of traces that the SUT should not be able to perform) is written in a more general format. Ideally, therefore, we would like to translate a finite automaton M, representing the traces that violate a safety property P, into a finite automaton M' such that σ' is a possible observation of some trace $\sigma \in L(M)$ ($\sigma \rightsquigarrow^* \sigma'$) if and only if $\sigma' \in L(M')$. However, we have the following impossibility result (a similar result was presented in our previous work for the case of general finite automata; here we show also that the result holds if we restrict attention to safety properties).

Proposition 1. *Given a safety property* P*, whose violation is represented by a finite automaton* M*, there may be no finite automaton* M' *with finite sets of states and transitions such that* $L(M') = \mathcal{L}(M)$*.*

Proof. Let us consider the formula $P = \Box((?i_1 \longrightarrow \bigcirc !o_1) \wedge (!o_1 \longrightarrow \bigcirc ?i_1))$ representing the requirement that input and output alternate. This is clearly a safety property and let M be the associated finite automaton that represents the minimal traces that violate P. Thus, traces in $L(M)$ have input and output alternating until some final suffix that has either two consecutive inputs or two consecutive outputs. We will use proof by contradiction, assuming that $\mathcal{L}(M)$ is a regular language. Thus, since $\mathcal{L}(M)$ is regular and $\{?i_1\}^*\{!o_1\}^*$ is regular we have that $\mathcal{L}(M) \cap \{?i_1\}^*\{!o_1\}^*$ is regular. However, this language contains all sequences of the form of n occurrences of the input $?i_1$ followed by m occurrences of output $!o_1$ such that n and m differ by at most 2 and this is not a regular language. This provides a contradiction as required. □

Thus, we cannot expect to have such an approach that deals with either all LTL formulae or all finite automata. In the following section we develop an alternative method where we construct finite automata from observed traces. Later, in Sections 4 and 5, we return to the problem of taking a property P represented

[2]In our previous work we used σ to refer to the trace defining a property. In this paper, we will use σ to refer to a trace of the SUT while ρ will be used for the former meaning

by finite automaton M and generating an appropriated version M' that takes into account asynchronous communications. Since we know that there is no general solution to this problem, we consider classes of finite automata, starting with acyclic automata. However, first we describe a more general approach.

3. A general method

Our previously developed approach [16] takes a property P represented by trace ρ and set O_ρ of outputs and generates a finite automaton M' such that $L(M') = \mathcal{L}(\rho(O \setminus O_\rho))$. The idea is that if the finite automaton accepts a sequence of actions, that is, a final state is reached, then it is possible that an erroneous behaviour was performed by the SUT (one that violated P). We have already seen that we cannot expect to always be able to apply such an approach when we use finite automata rather than traces to define properties (Proposition 1). However, there is an alternative general approach that can be applied for any finite automaton M that is based on defining a set $\mathcal{L}_r(\sigma)$ that is the set of traces that might have been produced by the SUT if we observe σ. Let us note that this can be seen as being exactly the opposite of our previous work: instead of considering traces that might be observed as variations of a trace $\sigma \in L(M)$ we consider the traces that might lead to the current observation. The challenge is to define a finite automaton $\mathcal{M}(\sigma)$ such that $L(\mathcal{M}(\sigma)) = \mathcal{L}_r(\sigma)$. If we can achieve this then the problem is one of deciding whether $L(\mathcal{M}(\sigma)) \cap L(M)$ is empty, a problem that can be decided in time that is polynomial in the size of M and $\mathcal{M}(\sigma)$. First we define $\mathcal{L}_r(\sigma)$.

Definition 3. *Given $\sigma \in \mathcal{A}ct^*$ we let $\mathcal{L}_r(\sigma) = \{\sigma' | \sigma' \rightsquigarrow^* \sigma\}$ denote the set of traces that can lead to the observation of σ. We will overload this to say that given an FA M, $\mathcal{L}_r(M) = \cup_{\sigma \in L(M)} \mathcal{L}_r(\sigma)$.*

Our previous approach was based on a finite automaton that recognised all traces that might be observed if the SUT produced a trace that contains ρ. As a result, we used $\rightsquigarrow$ to define the set of possible traces that might be observed if ρ was produced by the SUT (as part of a larger trace) and looked at the set of traces σ such that $\rho \rightsquigarrow^* \sigma$. Instead, in this alternative approach, we want to find the set of traces that *explain* an observation σ that has been made and these are the traces of the form σ' such that $\sigma' \rightsquigarrow^* \sigma$ (i.e. $\mathcal{L}_r(\sigma)$).

It might seem that given an observed trace σ we want to find a finite automaton M' such that $L(M') = \mathcal{L}_r(\sigma)$. However, we have to take into account the following factors:

1. It is sufficient for the trace of the SUT to contain a sub-trace σ' that is in $L(M)$ (recall that our properties are of the form $\Box P$).
2. Output produced by the SUT before such a sub-trace σ' might follow input from σ'.
3. Input from after the SUT produced such a sub-trace σ' might precede some output from σ'.

Thus, we will want $\mathcal{M}(\sigma)$ to denote the set of sequences σ' such that there exist $\sigma_1, \sigma_2 \in \mathcal{A}ct^*$ with $\sigma_1\sigma'\sigma_2 \in \mathcal{L}_r(\sigma)$.

Let us suppose that we have trace $\sigma = a_1 \dots a_k$ that has been observed by the monitor. Then we define the relation $\preceq$ on the inputs and outputs in σ, where $a_i \preceq a_j$ if the action corresponding to a_i *must* have been produced by the SUT no later than the action corresponding to a_j. We will want to be able to distinguish between two repeated occurrences of an input or an output in a trace σ and in the following we use $E(\sigma)$ to do this. The essential idea is that if σ contains k instances of a letter a then the first of these is represented by $(a, 1)$, the second by $(a, 2)$ etcetera.

Definition 4. *Let $\sigma = a_1 \dots a_n \in \mathcal{A}ct^*$ be a sequence of actions. We let $E(\sigma) \subseteq \mathcal{A}ct \times \mathbb{N}$ denote the* set of annotated actions of σ, *where $e = (a, k)$ belongs to $E(\sigma)$ if and only if there are k or more occurrences of a in σ. We define the function $pos_\sigma : E(\sigma) \longrightarrow \mathbb{N}$ such that for all $e = (a, k) \in E(\sigma)$ we have that $pos_\sigma(e) = i$ if $a = a_i$ and there are exactly $k - 1$ occurrences of a in $a_1 \dots a_{i-1}$. We will refer to the action e such that $pos_\sigma(e) = i$ by e_i. Abusing the notation we will write $e \in I$ (respectively, $e \in O$) if $e = (a, k)$ and $a \in I$ (respectively $a \in O$).*

Given $e_i, e_j \in E(\sigma)$ we write $e_i \preceq e_j$ if either $i = j$ or $i < j$ and one of the following conditions hold: e_i and e_j are both inputs, or e_i and e_j are both outputs, or e_i is an output and e_j is an input.

It is immediate that $\preceq$ is a partial order relation. The idea here is that we have $e_i \preceq e_j$ if the trace produced by the SUT must have had e_i before e_j. The first two cases come from communications being FIFO, while the third comes from the fact that if e_i is an output and is observed before an input e_j then the SUT must have produced e_i before it received e_j.

Throughout this paper we will not distinguish between an action $a_i \in \mathcal{A}ct$ and a corresponding value (a_i, k) in which a label has been added to make an observation (input or output) unique. An alternative would be to define a function from a labelled pair (a_i, k) to the corresponding action $a_i \in \mathcal{A}ct$; we chose not to do this since it would add little and would complicate the exposition.

Example 1. *Let us consider the trace $\sigma = ?i_1!o_1!o_2?i_2!o_1$. In order to distinguish between different occurrences of the same action, we label actions with the occurrence of that symbol in the trace. In this case, the corresponding set of actions is $E(\sigma) = \{(?i_1, 1), (!o_1, 1), (!o_2, 1), (?i_2, 1), (!o_1, 2)\}$. We have, for example, that $(?i_1, 1) \preceq (?i_2, 1)$, $(!o_2, 1) \preceq (!o_1, 2)$ and $(!o_1, 1) \preceq (?i_2, 1)$.*

We will base the construction of the finite automaton on the *ideals* of $E(\sigma)$ under the partial order $\preceq$.

Definition 5. *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions. A set $\mathcal{I} \subseteq E(\sigma)$ is an* ideal *of $(E(\sigma), \preceq)$ if for all $e_i, e_j \in E(\sigma)$, if $e_i \preceq e_j$ and $e_j \in \mathcal{I}$ then $e_i \in \mathcal{I}$. Further, $E' \subseteq E(\sigma)$ is an* anti-chain *of $(E(\sigma), \preceq)$ if no two different elements of E' are related under $\preceq$.*

It is straightforward to see that a set $\mathcal{I}$ is an ideal if and only if one of the following conditions holds:

- $\mathcal{I}$ contains an output a_i and all earlier outputs;
- $\mathcal{I}$ contains an input a_j and all earlier inputs and outputs; or
- $\mathcal{I}$ contains an input a_i, an output a_j, all inputs and outputs before a_i, and all outputs before a_j.

Example 2. *Let us consider again the trace* $\sigma = ?i_1!o_1!o_2?i_2!o_1$. *The ideals of* $(E(\sigma), \preceq)$ *can be grouped into the three previously defined categories as follows:*

- *Output and previous outputs:*
  $\{(!\mathbf{o_1}, 1)\}$, $\{(!o_1, 1), (!\mathbf{o_2}, 1)\}$ *and* $\{(!o_1, 1), (!o_2, 1), (!\mathbf{o_1}, 2)\}$.
- *Input and previous actions:*
  $\{(?\mathbf{i_1}, 1)\}$ *and* $\{(?i_1, 1), (!o_1, 1), (!o_2, 1), (?\mathbf{i_2}, 1)\}$.
- *Input and output; actions previous to the input and outputs previous to the output:*
  $\{(?\mathbf{i_1}, 1), (!\mathbf{o_1}, 1)\}$, $\{(?\mathbf{i_1}, 1), (!o_1, 1), (!\mathbf{o_2}, 1)\}$, $\{(?\mathbf{i_1}, 1), (!o_1, 1), (!o_2, 1), (!\mathbf{o_1}, 2)\}$ *and* $\{(?i_1, 1), (!o_1, 1), (!o_2, 1), (?\mathbf{i_1}, 2), (!\mathbf{o_1}, 2)\}$.

  In addition, we have the empty ideal $\{\}$.

Using a classical result [13], relating ideals and anti-chains, we have that the number of ideals is at worst quadratic in the size of σ. Even though we use a different partial order, the proof follows the same lines as that in our previous work [16].

Proposition 2. *Let* $\sigma \in \mathcal{A}ct^*$ *be a sequence of actions with length* m. *There are* $O(m^2)$ *ideals of* $(E(\sigma), \preceq)$.

Let us note that there exists an interesting connection between the ideals generated from a trace and the partial order $\preceq$ and the prefixes of the trace. Actually, each ideal corresponds to a different set of prefixes of the possible real trace produced by the SUT. For instance, in the previous example, the ideal $\{(?i_1, 1), (!o_1, 1), (!o_2, 1), (!o_1, 2)\}$ corresponds to the situation where, even though we observed the trace $\sigma = ?i_1!o_1!o_2?i_2!o_1$, the SUT originally produced a trace such as $\sigma = ?i_1!o_1!o_2!o_1?i_2$ but the observation of the last output was delayed. This relation is explicitly shown in the construction of the desired finite automaton.

Definition 6. *Let* $\sigma \in \mathcal{A}ct^*$ *be a non-empty trace. We let* $\mathcal{M}'(\sigma)$ *denote the finite automaton with state set equal to the set of ideals of* $(E(\sigma), \preceq)$, *alphabet* $\mathcal{A}ct$, *initial state* $\{\}$ *and the following set of transitions: given ideal* $\mathcal{I}$ *and* $a \in E(\sigma)$, *there is a transition* $t = (\mathcal{I}, a, \mathcal{I}')$ *for ideal* $\mathcal{I}'$ *if and only if* $\mathcal{I}' = \mathcal{I} \cup \{a\}$. *In addition,* $\mathcal{M}'(\sigma)$ *has one final state, which is the ideal including all the actions of the trace, that is,* $E(\sigma)$.

Algorithm 1 Producing $\mathcal{M}(\sigma)$

1: Input σ.
2: Let $\mathcal{M}(\sigma) = \mathcal{M}'(\sigma)$,
3: Let s_0 denote the initial state of $\mathcal{M}(\sigma)$. For every state s, add to $\mathcal{M}(\sigma)$ the transition (s_0, ϵ, s).
4: Make every state of $\mathcal{M}(\sigma)$ a final state.
5: Output $\mathcal{M}(\sigma)$.

The following is immediate from the definition of $\mathcal{M}'(\sigma)$.

Proposition 3. *Let $\sigma \in \mathcal{A}ct^*$ be a non-empty trace observed through interacting with the SUT via an FIFO asynchronous channel. We have that σ' is a trace that the SUT might have actually produced if and only if $\sigma' \in L(\mathcal{M}'(\sigma))$.*

We now adapt $\mathcal{M}'(\sigma)$ to take into account the points discussed: it is sufficient that the SUT produced a trace $\sigma_1\sigma'\sigma_2$ for some σ' in $L(\mathcal{M}'(\sigma))$. The paths of $\mathcal{M}'(\sigma)$ define the possible traces that explain the observation of σ and now it is sufficient to adapt $\mathcal{M}'(\sigma)$ so that the new finite automaton accepts all subtraces of the traces of $\mathcal{M}'(\sigma)$. To allow the removal of prefixes we simply add transitions with label ϵ from the initial state of $\mathcal{M}'(\sigma)$ to each state of $\mathcal{M}'(\sigma)$. To allow the removal of suffixes of traces of $\mathcal{M}'(\sigma)$ we simply make every state a final state. Algorithm 1 shows how the finite automaton $\mathcal{M}(\sigma)$ is built. As explained earlier, we overload ϵ to denote both an empty trace (Definition 1) and an empty label in a finite automaton (Algorithm 1).

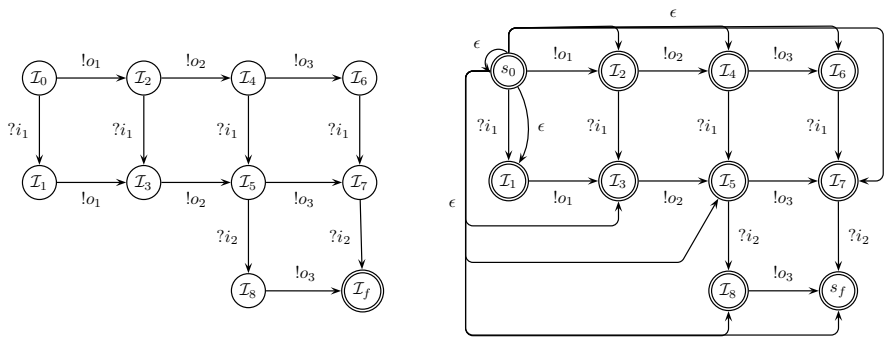
Example 3. *Let $\sigma = ?i_1!o_1!o_2?i_2!o_3$ be a trace. Figure 3 depicts the finite automata $\mathcal{M}'(\sigma)$ (left) and $\mathcal{M}(\sigma)$ (right).*

The next result shows that our finite automaton satisfies the required property.

Proposition 4. *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions. If trace σ is observed through interacting with the SUT via an FIFO asynchronous channel then $\sigma' \in L(\mathcal{M}(\sigma))$ if and only if there exist σ_1, σ_2 such that $\sigma_1\sigma'\sigma_2 \in L(\mathcal{M}'(\sigma))$.*

Proof. We first prove the left to right implication. If $\sigma' \in L(\mathcal{M}(\sigma))$ then there exist states s and s' of $\mathcal{M}(\sigma)$ such that $s_0 \xrightarrow{\epsilon} s \stackrel{\sigma'}{\Longrightarrow} s'$. Given the state s of $\mathcal{M}'(\sigma)$ there exists a trace σ_s that leads the finite automaton from its initial state to s without using transitions with label ϵ because $\mathcal{M}'(\sigma)$ is connected by construction. We set $\sigma_1 = \sigma_s$. In addition, by the construction of $\mathcal{M}'(\sigma)$, the final state can be reached from any state of the finite automaton. We can set σ_2 to be a trace that labels a walk from state s' to the final state of $\mathcal{M}'(\sigma)$.

Now we will prove the right to left implication. Since $\sigma_1\sigma'\sigma_2 \in L(\mathcal{M}'(\sigma))$ there exist states s, s' such that $s_0 \stackrel{\sigma_1}{\Longrightarrow} s \stackrel{\sigma'}{\Longrightarrow} s'$. By the construction of $\mathcal{M}(\sigma)$ there exists a transition labelled by ϵ that can reach state s from the initial state of $\mathcal{M}(\sigma)$. Next, since the states of $\mathcal{M}(\sigma)$ and $\mathcal{M}'(\sigma)$ are the same, the trace σ'



$$
\begin{array}{lcllcl}
\mathcal{I}_0 & = & \{\} & \mathcal{I}_5 & = & \{?i_1, !o_1, !o_2\} \\
\mathcal{I}_1 & = & \{?i_1\} & \mathcal{I}_6 & = & \{!o_1, !o_2, !o_3\} \\
\mathcal{I}_2 & = & \{!o_1\} & \mathcal{I}_7 & = & \{?i_1, !o_1, !o_2, !o_3\} \\
\mathcal{I}_3 & = & \{?i_1, !o_1\} & \mathcal{I}_8 & = & \{?i_1, !o_1, !o_2, ?i_2\} \\
\mathcal{I}_4 & = & \{!o_1, !o_2\} & \mathcal{I}_9 & = & \{?i_1, !o_1, !o_2, ?i_2, !o_3\}
\end{array}
$$

Figure 3: Finite automata $\mathcal{M}'(\sigma)$ (left) and $\mathcal{M}(\sigma)$ (right) for the trace $\sigma = ?i_1!o_1!o_2?i_2!o_3$

can be performed in $\mathcal{M}(\sigma)$ from the state s and we reach a final state because all the states of $\mathcal{M}(\sigma)$ are final. Therefore, $\sigma' \in L(\mathcal{M}(\sigma))$. □

The following gives corresponding computational complexity results.

Proposition 5. *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions. The number of states of the finite automaton $\mathcal{M}(\sigma)$ is of $O(|\sigma|^2)$. In addition, the process of checking whether a trace σ' is accepted by $\mathcal{M}(\sigma)$ is of $O(|\sigma|^2 \cdot |\sigma'|)$.*

Proof. The first part is immediate from Proposition 2 and the definition of $\mathcal{M}(\sigma)$.

Now consider the problem of deciding whether σ' is accepted by $\mathcal{M}(\sigma)$. For each state s of $\mathcal{M}(\sigma)$ we have a corresponding membership problem: that of deciding whether σ' is in the language defined by the deterministic finite automaton formed from $\mathcal{M}(\sigma)$ by making s the initial state and removing all of the ε transition. We therefore have $O(|\sigma|^2)$ membership problems for deterministic finite automata. But the membership problem for a deterministic finite automaton can be solved in time that is linear in terms of the length of the sequence σ'. Thus, the problem of deciding whether σ' is accepted by $\mathcal{M}(\sigma)$ is of $O(|\sigma|^2 \cdot |\sigma'|)$. □

The outlined approach takes low-order polynomial time. However, it is polynomial in terms of the length of the trace σ observed and this may be relatively large. The effect can be reduced by building $\mathcal{M}(\sigma)$ in an incremental manner: we build $\mathcal{M}(\sigma a)$ from $\mathcal{M}(\sigma)$. We now explain how this can be achieved.

Let us suppose that we observed trace σ, have built $\mathcal{M}(\sigma)$, and have now made an additional observation a. There are two cases to consider.

Algorithm 2 Producing $\mathcal{M}(\sigma a)$

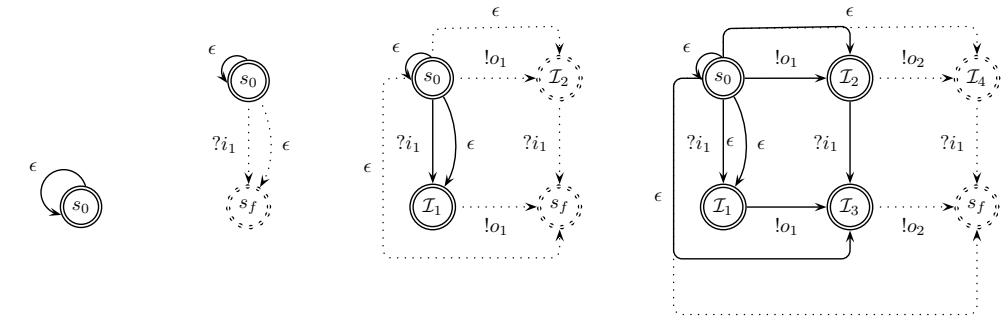
1: Input $\mathcal{M}(\sigma) = (S, \mathcal{A}ct, Tr, s_0, S)$ and action a.
2: Let $\mathcal{M}(\sigma a) = \mathcal{M}'(\sigma)$.
3: Let $s_f \in S$ be the state in $\mathcal{M}(\sigma)$ representing $E(\sigma)$.
4: If $a \in I$, add a state s representing $E(\sigma a)$ and the transitions (s_f, a, s) and (s_0, ϵ, s).
5: If $a \in O$, for every state $s_{\mathcal{I}}$ that represents an ideal $\mathcal{I}$ that contains all of the outputs from σ, add a state s' representing ideal $\mathcal{I} \cup \{a\}$ and the transitions $(s_{\mathcal{I}}, a, s')$ and (s_0, ϵ, s'). In addition, for every action a' labeling a transition outgoing from $s_{\mathcal{I}}$ we add a transition (s', a', s'') where s'' represents the ideal $\mathcal{I} \cup \{a, a'\}$.
6: Make all the new states final.

1. Action a is an input. Then a will be received after all actions from σ and so we simply add a new state s representing $E(\sigma a)$, make this a final state, add a transition from the state representing $E(\sigma)$ to s with label a, and add the transition (s_0, ϵ, s).
2. Action a is an output. Then a was produced after all outputs from σ but might have been produced before inputs that appear in σ. For each state s_1 represented by an ideal $\mathcal{I}$ that contains all outputs from σ we add a state s_1' representing ideal $\mathcal{I} \cup \{a\}$ and a transition from s_1 to s_1'. For all a' such that there is a state with ideal $\mathcal{I} \cup \{a'\}$ we add a transition from s_1' with action a' to the state representing $\mathcal{I} \cup \{a, a'\}$. We make all states final states and add transitions with label ϵ from s_0 to the new states.

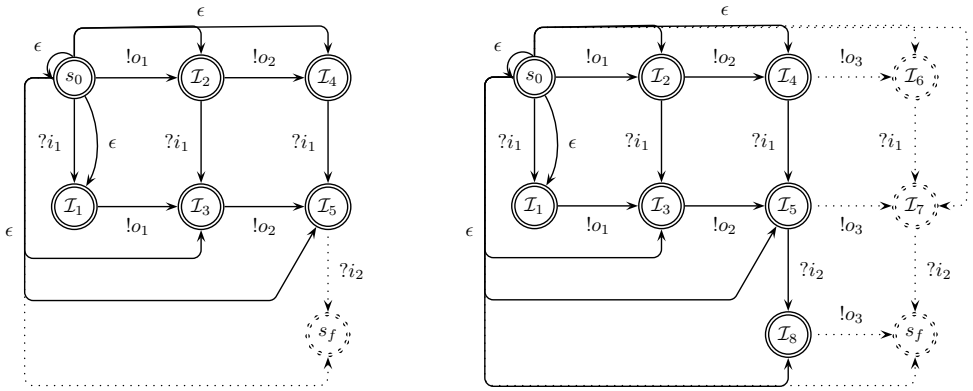
Proposition 6. *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $a \in \mathcal{A}ct$. Given the finite automaton $\mathcal{M}(\sigma)$, the process of generating the finite automaton $\mathcal{M}(\sigma a)$ is of $O(|\sigma|)$ time.*

Proof. If a is an input then we simply add a new state. If a is an output, then the algorithm includes as many new states as ideals containing all the outputs in σ, that is, in the worst case we add $|\sigma|$ states. In addition, a new transition is included in the finite automaton for each new state. Finally, for each transition outgoing from ideals containing all the outputs in σ, a new transition is added. By the construction of the finite automaton, there is at most one transition (labelled by an input action) from each of these states. Therefore, in the worst case the algorithm performs $3 \cdot |\sigma|$ operations and the overall time complexity is therefore of $O(|\sigma|)$. □

Example 4. *Let us consider the observation of the trace $\sigma = ?i_1!o_1!o_2?i_2!o_3$. Figure 4 shows how the finite automaton $\mathcal{M}(\sigma)$ is built in an incremental way. We present with dashed circles/lines the states/transitions that are added at each step. Let us assume that we have built the finite automaton $\mathcal{M}(?i_1!o_1!o_2)$ and the action $?i_2$ is observed. In this case, the algorithm only adds a new state and a transition labelled with the action $?i_2$. It is not possible for an input to*



(a) ϵ (b) $?i_1$ (c) $?i_1!o_1$ (d) $?i_1!o_1!o_2$



(e) $?i_1!o_1!o_2?i_2$ (f) $?i_1!o_1!o_2?i_2!o_3$

Figure 4: Construction of finite automaton $\mathcal{M}(\sigma)$ by Algorithm 2 for the trace $\sigma = ?i_1!o_1!o_2?i_2!o_3$

be received by the SUT before it has been observed or before the actions that have been observed previously. Therefore, it must be the last action in any trace produced by the SUT that can lead to the observation of $?i_1!o_1!o_2?i_2$. *The last transformation of the finite automaton corresponds to the observation of the action* $!o_3$. *Due to the fact that we are considering an FIFO asynchronous setting,* $!o_3$ *must have been produced after all the outputs observed before it. However, it is possible that the observation was produced by the SUT before some of the input actions observed earlier and it has been observed later due to a delay. Thus, the algorithm extends the finite automaton with new states and transitions that reflect those issues. First, new transitions and states that capture the observation of* $!o_3$ *after the outputs* $!o_1$ *and* $!o_2$ *are included in the finite automaton:* $(\mathcal{I}_4, !o_3, \mathcal{I}_6), (\mathcal{I}_5, !o_3, \mathcal{I}_7)$ *and* $(\mathcal{I}_8, !o_3, s_f)$. *Second, new transitions are added to reflect a possible delay in the observation of* $!o_3$ *with respect to the inputs observed previously:* $(\mathcal{I}_6, ?i_1, \mathcal{I}_7), (\mathcal{I}_7, ?i_2, s_f)$.

Finally, we have the following result that the overall process takes polynomial

time. The the result follows immediately from it being possible to construct $\mathcal{L}_r(\sigma)$ in polynomial time and it is possible to decide whether the intersection of two finite automata defines a non-empty language in polynomial time.

Theorem 1. *Let $\sigma \in \mathcal{A}ct^*$ be the observed trace and M be a finite automaton representing a property that we are checking. It is possible to decide in time that is polynomial in the sizes of σ and M whether an erroneous behaviour might have occurred.*

The approach described in this section has the advantages that it is a polynomial time method that works with any property expressed as a finite automaton. This represents significant progress with respect to our previous work where we were restricted to properties of the form "if we observe a particular sequence of actions then the next observed output must belong to a certain set of outputs". However, the disadvantage is that it generates a finite automaton $\mathcal{M}(\sigma)$ whose size is quadratic in the length of σ. In contrast, the space required by our previous approach did not depend on the length of the observed trace. In practice, one expects σ to be much larger than the finite automaton M representing the violation of the property of interest and so it is desirable to have approaches whose space complexity does not depend on σ. However, there may be situations in which dependence on the size of σ is not so problematic since there are frequent actions such as a session ending or a disconnect that effectively reset the trace. We now consider conditions under which it is possible to devise an approach that rewrites M rather than σ.

4. Acyclic finite automata

In this section we consider the case where the property of interest can be represented by an acyclic finite automaton M. The problem then is to transform M into a finite automaton M' such that (there exists $\sigma \in L(M)$ such that $\sigma \rightsquigarrow^* \sigma'$) if and only if $\sigma' \in L(M')$. We show how this can be done in a manner that can take space and time that is exponential in the size of M before showing that it is not generally possible to avoid a combinatorial explosion.

Our previous approach [16] defined a partial order $\ll$ on the set $E(\sigma)$ of actions in a trace $\sigma = a_1 \ldots a_n$, where $E(\sigma) = \{e(a_1), \ldots, e(a_n)\}$ is the set of actions in σ labelled in order to make them unique. This was defined by the following.

Definition 7. *Let $\sigma = a_1 \ldots a_n \in \mathcal{A}ct^*$ be a sequence of actions. Given two actions $e_i = (a_i, k_i)$ and $e_j = (a_j, k_j)$ belonging to $E(\sigma)$, we write $e_i \ll e_j$ if either $i = j$ or $i < j$ and one of the following conditions holds: a_i and a_j are inputs, or a_i and a_j are outputs, or a_i is an input and a_j is an output.*

The idea is that if $a_i \ll a_j$ then in any σ' with $\sigma \rightsquigarrow^* \sigma'$ we have that a_i must appear before a_j in σ'. Then we have that $\sigma \rightsquigarrow^* \sigma'$ if and only if the ordering of the actions in $E(\sigma')$ satisfies the partial order $\ll$ and so $\mathcal{L}(\sigma)$ is the

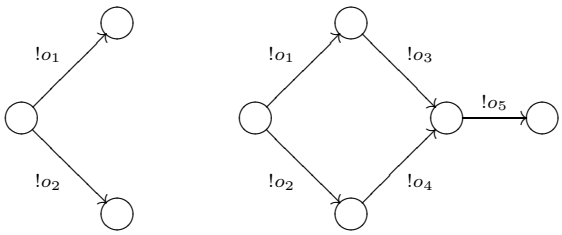


Figure 5: An FA where an ideal of $\ll$ does not define a trace (left) and an FA where a trace does not define an ideal of $\ll$ (right)

set of linearisations of $E(\sigma)$ under partial order $\ll$ (once labels are removed from actions). We then defined a finite automaton in terms of the ideals of $E(\sigma)$.

In this section we consider the case where M is defined by an acyclic finite automaton rather than a single trace σ. The following defines a natural extension of the partial order on actions.

Definition 8. *Let $M = (Q, \mathcal{A}ct, T, q_{in}, F)$ be an acyclic FA, with actions on transitions relabelled to make them unique if necessary, and let $E(M)$ denote the set of actions of M. Given two actions a_i, a_j of M we write $a_i \ll_M a_j$ if M contains a walk whose label starts with a_i and ends with a_j and one of the following conditions hold: a_i and a_j are inputs, or a_i and a_j are outputs, or a_i is an input and a_j is an output.*

The following example shows that, in contrast to our previous work, in this more complex setting it is not sufficient to consider ideals of $\ll_M$.

Example 5. *Let us consider the acyclic FA M shown in Figure 5 (left), in which all states are final states. There are two actions, both with outputs, and clearly these would be unrelated under $\ll_M$ if we based $\ll_M$ on the traces. Thus, the set $\{!o_1, !o_2\}$ is an ideal but clearly no trace of M contains these actions.*

This shows that an ideal need not correspond to a trace of M and the following shows that the set of actions in a trace of M need not form an ideal.

Example 6. *Consider the acyclic FA M shown in Figure 5 (right), in which all states are final states, and the trace $\sigma = !o_1!o_3!o_5$. Then it is straightforward to see that the set $E(\sigma)$ of actions in this trace does not form an ideal under $\ll_M$ since $E(\sigma)$ contains $!o_5$, we have that $!o_2 \ll_M !o_5$, but $E(\sigma)$ does not contain $!o_2$.*

Thus, it appears that our previous approach cannot be extended in the natural way and we have to look for an alternative solution.

Since M is acyclic, $L(M)$ is finite. Thus, we can produce all traces from $L(M)$, for each such trace σ produce a finite automaton $\mathcal{M}_O(\sigma)$ (using our previous approach [16]), and define a finite automaton that accepts the union, over $\sigma \in L(M)$, of the $\mathcal{M}_O(\sigma)$. This approach is summarised in Algorithm 3.

Algorithm 3 Producing $\mathcal{M}_O(\sigma)$

1: Input M.
2: Let M' denote a finite automaton defining the empty language.
3: **for all** $\sigma \in L(M)$ **do**
4: Produce $\mathcal{M}_O(\sigma)$
5: Update M' by taking the disjoint union with $\mathcal{M}_O(\sigma)$ and then identifying their initial states.
6: **end for**
7: Output M'.

Correctness immediately follows from the previously defined construction (for a trace σ) being correct [16].

Theorem 2. *Given acyclic finite automaton M, if M' is the finite automaton returned by Algorithm 3 then $L(M') = \mathcal{L}(M)$.*

We know from previous work [16] that given trace σ, we can devise $\mathcal{M}_O(\sigma)$ in time of $O(|\sigma|^2)$ and this has at most $|\sigma|^2$ states. In addition, since M is acyclic we have that the length of σ is bounded above by the number of states of M minus 1 and so $\mathcal{M}_O(\sigma)$ can be devised in time that is polynomial in the size of M. Given $\sigma, \sigma' \in L(M)$ we can define a finite automaton that accepts $L(\mathcal{M}_O(\sigma)) \cup L(\mathcal{M}_O(\sigma'))$ by merging the initial states of $\mathcal{M}_O(\sigma)$ and $\mathcal{M}_O(\sigma')$ and so the overall complexity of the approach is polynomial in the size of M and linear in the number of traces in $L(M)$. This is summarised by the following.

Proposition 7. *Given acyclic finite automaton M with n states, the time complexity of Algorithm 3 is of $O(|L(M)|n^2)$. In addition, if M' is the finite automaton returned by Algorithm 3 then M' has at most $|L(M)|(n-1)^2$ states.*

However, the number of traces in $L(M)$ can grow exponentially as the number of states of M increases and so the overall approach can take space that is exponential in terms of the size of M. The following result shows that we cannot avoid a combinatorial explosion.

Proposition 8. *There exists a class of acyclic finite automata $M_1, M_2, \ldots$ such that the number of states required by a finite automaton that represents $\mathcal{L}(M_k)$ is exponential in k.*

Proof. For $k > 0$ consider the FA M_k shown in Figure 6 in which the state with no outgoing transitions is the unique final state. Let us suppose that M'_k is an FA such that $L(M'_k) = \mathcal{L}(M_k)$. Let I_1 and I_2 denote two different subsets of $\{?i_1, \ldots, ?i_k\}$ and let us consider traces σ_1 and σ_2 such that σ_1 (respectively σ_2) contains exactly the actions in I_1 (respectively I_2) and in the order specified in M_k. Then σ_1 and σ_2 are clearly prefixes of traces of $\mathcal{L}(M_k)$; traces in which the outputs have been delayed past these initial inputs. In addition, since $I_1 \neq I_2$, the set of traces that can follow σ_1 and σ_2 in $\mathcal{L}(M_k)$ differ. Thus, we must have that σ_1 and σ_2 reach different states of M'_k. However, this holds for any

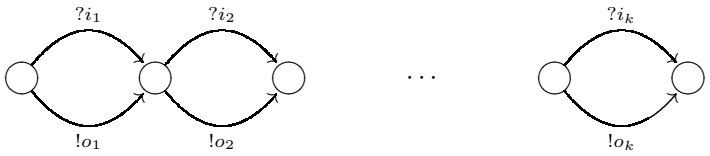


Figure 6: An FA M_k such that $\mathcal{L}(M)$ requires exponentially many states

distinct subsets I_1 and I_2 of $\{?i_1, \ldots, ?i_k\}$ and so each subset of $\{?i_1, \ldots, ?i_k\}$ must map to a corresponding state of M'_k. As a result, since there are 2^k subsets of $\{?i_1, \ldots, ?i_k\}$, M'_k must have at least 2^k states. □

While the complexity is exponential in terms of the number of states of M, this is exponential in terms of the size of the property. Having generated M', the process of monitoring is linear in terms of the length of the trace σ observed. Since M will typically be much smaller than σ, in some cases this may be preferable to the approach outlined in Section 3 in which the complexity is polynomial in terms of the sizes of M and σ but is quadratic in terms of the size of σ. Future work will consider conditions under which it is possible to produce a finite automaton that recognises $\mathcal{L}(M)$ in polynomial time.

5. Allowing cycles

In this section we consider the case where a property is defined by a finite automaton that contains cycles. Mazurkiewicz trace theory uses the notion of a *partial commutation*, which is based on an independence relation Ind. Under this, actions a and b being independent ($(a, b) \in Ind$) denotes them commuting and so ab and ba being equivalent (see, for example, [24]). The independence relation in partial commutations is symmetric but in exploring asynchronous communications we require a notion of independence that is asymmetric, since output can be delayed but input cannot. This corresponds to a *semi-commutation* in which there is an independence relation Ind on pairs of actions from $\mathcal{A}ct$: $(a, b) \in Ind$ if and only if $a \in O$ and $b \in I$. Given an independence relation Ind we also have a dependence relation D defined by $(a, b) \in D$ if and only if $(a, b) \notin Ind$. In the context of the work in this paper, an input cannot be delayed. In addition, since communications are FIFO, one output cannot be delayed beyond a previous output. Thus, the dependence relation is: $D = \{(a, b) \in \mathcal{A}ct \times \mathcal{A}ct | a \in I \vee (a \in O \wedge b \in O)\}$.

Let us suppose that we have a semi-commutation with dependence relation D. Given trace σ we can define a directed graph $G_{\sigma,D}$ on the letters in σ as follows: if a and b both appear in σ then there is an edge from a to b if and only if $(a, b) \in D$. A trace σ is said to be *strongly-connected* if the directed graph $G_{\sigma,D}$ is strongly connected (for every ordered pair (v, v') of vertices of $G_{\sigma,D}$

there is a path in $G_{\sigma,D}$ from v to v') [11]. The following is immediate from the definition of independence for FIFO communications.

Proposition 9. *A trace $\sigma \in \mathcal{A}ct^*$ is strongly-connected if and only if either it only contains inputs or it only contains outputs.*

It is known ([11], Proposition 6.17) that if every cycle of a finite automaton M is strongly-connected then the corresponding language $\mathcal{L}(M)$ is regular. We therefore obtain the following.

Proposition 10. *Given a finite automaton M, if every cycle of M either contains only inputs or contains only outputs then $\mathcal{L}(M)$ is a regular language.*

Thus, this provides a sufficient condition for there to be a finite automaton M' such that $L(M') = \mathcal{L}(M)$. The following defines this class of finite automata.

Definition 9. *An FA M is a Consistent FA if whenever σ is the label of a cycle of M we have that either $\sigma \in I^*$ or $\sigma \in O^*$.*

In order to represent the language $\mathcal{L}(M)$ by a finite automaton M' we require that this language is regular. Thus, an FA M being a Consistent FA is a condition under which there must be such a finite automaton M'. Throughout this section we assume that the FA M considered has the following properties.

Assumption 1.

M is a Consistent FA.

All states of M can be reached from the initial state of M and for every state s of M there is a path from s to a final state of M.

The first of these assumptions is simply the condition above. Making the second assumption will simplify the analysis in this section but does not affect the generality of the work: if M does not satisfy this second condition then we can rewrite it to form an equivalent FA that does satisfy this condition. It is straightforward to check whether FA M satisfies these conditions.

Proposition 11. *Given a finite automaton M with k transitions, it is possible to check whether M satisfies Assumption 1 in time of $O(k^3)$.*

Proof. First consider the problem of deciding whether M is a Consistent FA. For this, it is sufficient to check whether there is a cycle that contains both inputs and outputs. To decide this, for every pair t, t' of transitions of M such that the action in t is an input and the action in a' is an output, we check whether M has a cycle that contains t and t'. There is such a cycle if and only if there is a path from the end state of t to the start state of t' and also a path from the end state of t' to the start state of t. Thus, it is sufficient to check the $O(k^2)$ such pairs of transitions and each check can be achieved in $O(k)$ time using a depth-first search. It is therefore possible to check that M is a Consistent FA in time of $O(k^3)$.

To check the second condition, for each state s it is sufficient to determine whether s can be reached from the initial state and also whether there is a path from s to some final state. For a state s the first part can be solved in $O(k)$ time using a depth-first search and the second part can also be solved in $O(k)$ time using a depth-first search. Since there are $O(k)$ states, it is possible to check the second condition in $O(k^2)$ time. □

The challenge now is to devise an algorithm for constructing a finite automaton whose language is $\mathcal{L}(M)$, where M is a Consistent FA.

First we will consider the structure of a Consistent FA M. Every walk of M to a final state is in the form of a path ρ with cycles attached. As a result, it is sufficient to consider each (acyclic) path ρ that takes M to a final state and all cycles that can be taken from states of ρ; such a path ρ and its associated cycles will form an FA M_ρ. Later we will see how we can construct an FA $\mathcal{M}_C(M_\rho)$ such that $L(\mathcal{M}_C(M_\rho)) = \mathcal{L}(M_\rho)$; the final FA will be formed by combining the $\mathcal{M}_C(M_\rho)$.

In order to find such a path ρ and associated transitions we can employ the following approach. This leads to a set of finite automata of the form M_ρ.

1. We take each path ρ to a final state s of M (ρ might contain both inputs and outputs but might also contain only inputs or only outputs). This is the same as the process described for the acyclic case.
2. Given ρ, we add every transition t that is in a cycle with a state of ρ. To achieve this, for state s of ρ and a transition t of M with start state s' and end state s'', we determine whether there is a path from s to s' and a path from s'' to s.

Proposition 12. *Given a finite automaton M with k transitions and path ρ that takes M to a final state, it is possible to devise M_ρ in time of $O(k^3)$.*

Proof. Given state s and transition t it is sufficient to decide whether there is a cycle that contains s and t. There is a such a cycle if and only if there is a path from the end state of t to s and also a path from s to the start state of t. Thus, it is sufficient to check the $O(k)$ such transitions and, for a given transition t, this can be achieved in $O(k)$ time using a depth-first search. This process, for state s, thus takes time of $O(k^2)$. Finally, observe that since ρ is a path it has no repeated states and so it has $O(k)$ states. The result therefore follows. □

We now consider how one can reason about such a finite automaton M_ρ. The proposed approach will be based on the following steps.

1. Make the inputs and outputs unique; as before we add unique labels.
2. We use pairs of the form (a, b) to represent states of the new FA that recognises $\mathcal{L}(M_\rho)$ where a is a labelled input and b is a labelled output. The pair (a, b) denotes the situation in which the most recently observed input was a and the most recently observed output was b.
3. We carefully construct the transitions between these states by determining what inputs and outputs can next be observed.

In order to uniquely label an action in the automaton, we will simply use the name of the corresponding transition (Definition 10). Next, we define a relation that determines the pairs of transitions that may correspond to the last observed input and last observed output. In this we need to take into account the nature of communications and the resultant possibility of delaying an output. We also include a transition $(s_{in}, \epsilon, s_{in})$ to represent the (initial) situation in which no observations have been made.

Definition 10. *Let M be a Consistent FA and let ρ be a path that takes M to a final state. We let $Tr(M_\rho)$ denote the* set of transitions of M_ρ, *where $tr = (s_a, a, s'_a)$ belongs to $Tr(M_\rho)$ if and only if there exists a transition in M_ρ with start state s_a, end state s'_a and label a. We will write $tr \in Tr_I(M_\rho)$ (respectively, $tr \in Tr_O(M_\rho)$) if $tr = (s_a, a, s'_a)$ and $a \in I$ (respectively $a \in O$). We will include $(s_{in}, \epsilon, s_{in})$ in $Tr(M_\rho)$, where s_{in} is the initial state of M.*

We define the observational relation between transitions as follows. Let $tr_\alpha = (s_\alpha, a, s'_\alpha)$ and $tr_\beta = (s_\beta, b, s'_\beta)$ belong to $Tr(M_\rho)$. We write $tr_\alpha \bowtie_{M_\rho} tr_\beta$ if and only if $a \in I \cup \{\epsilon\}$ and $b \in O \cup \{\epsilon\}$ and one of the following conditions hold:

- *$s'_\alpha = s_\beta \vee s'_\beta = s_\alpha$. This is simply the case where we have two consecutive transitions.*
- *There exists a path $(s'_\alpha, a_1, s_1) \ldots (s_{n-1}, a_n, s_\beta)$ of M_ρ with $n > 0$ and for all $1 \leq k \leq n$ we have that $a_k \in O$. In this case, even though the two transitions tr_α and tr_β are not consecutive, there are no additional inputs between tr_α and tr_β in the path and so when tr_β has been executed we have that tr_α is still the most recently executed transition whose label is an input.*
- *There exists a path $(s'_\beta, a_1, s_1) \ldots (s_{n-1}, a_n, s_\alpha)$ in M_ρ with $n > 0$. This is a possible pair of most recent input/output transitions since it is possible for all output after that from (s_β, b, s'_β) to be delayed beyond the observation of a.*

We denote by $Obs(M_\rho)$ the set of pairs (a, b) such that $a \bowtie_{M_\rho} b$.

Figure 7 illustrates the observational relation between an input and an output action with respect to the depicted automaton. The dotted arcs represent the possible observations of output b. We have that $a \bowtie_{M_\rho} b$, $a \bowtie_{M_\rho} b'$ $a' \bowtie_{M_\rho} b$, $a' \bowtie_{M_\rho} b'$ and $a' \bowtie_{M_\rho} b''$ while $a \bowtie_{M_\rho} b''$ does not hold. This is because all the paths from the initial state to state s_3 all traverse transition (s_2, a', s_3) and so it is not possible to be in the situation in which we have observed action b'' and action a' has not been received by the SUT. Therefore, if the last observed output is b'' then we must have that the last observed input is a'. The second case in the definition of the relation captures this situation. The fact that outputs can be delayed leads to output b being related to input a'.

We have that the number of pairs is at worst quadratic in the number of transitions of M_ρ.

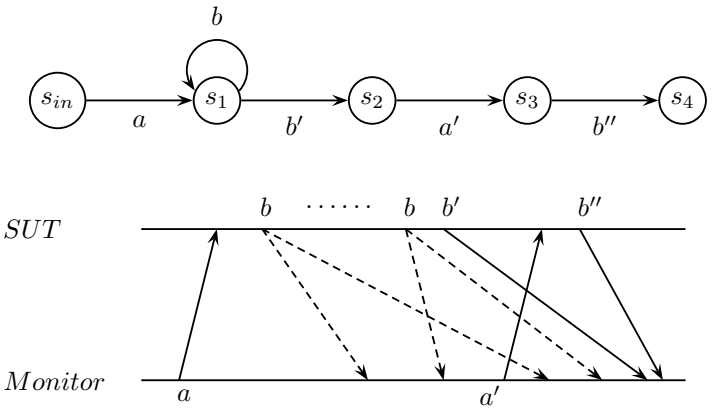


Figure 7: Observational relation

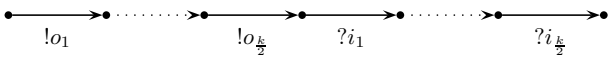


Figure 8: Finite automaton such that $|Obs(M_\rho)| = \frac{k^2}{4}$ elements.

Proposition 13. *Let M be a Consistent FA and ρ a path of M that takes M to a final state. If M_ρ has k transitions then there are $O(k^2)$ pairs in $Obs(M_\rho)$.*

Proof. This follows from each pair in $Obs(M_\rho)$ representing a pair of transitions of M_ρ and their being $O(k^2)$ such pairs. □

Consider now the finite automaton depicted in Figure 8. In this case all transitions labelled with an input action are related to all transitions labelled with an output action and so we have $\frac{k^2}{4}$ elements in $Obs(M_\rho)$. This demonstrates that the complexity cannot be reduced below quadratic. We now consider the complexity of finding the relation $\bowtie_{M_\rho}$.

Proposition 14. *Let M be a Consistent FA and ρ a path of M that takes M to a final state. If M_ρ has k transitions then the process of finding the pairs of transitions related under $\bowtie_{M_\rho}$ is of $O(k^2)$.*

Proof. Given transition t it is sufficient to find every transition t' such that there exists a path from the end state of t to the initial state of t' fulfilling the conditions imposed in Definition 10. This can be achieved in $O(k)$ time using a depth-first search [29]. Since there are k transitions in M_ρ, the process of generating $Obs(M_\rho)$ is of $O(k^2)$ time. The result therefore follows. □

In the following, we show how the pairs in $Obs(M_\rho)$ can be used to construct an appropriate automaton. Specifically, we will use the pairs of $Obs(M_\rho)$ to represent states of a finite automaton that accepts the set of sequences in $\mathcal{L}(M_\rho)$. Recall that in computing $Obs(M_\rho)$ we include the transition $(s_{in}, \epsilon, s_{in})$, which allows us to have states that represent the situation where no inputs and/or outputs have been observed.

Definition 11. *Given Consistent FA M and path ρ of M that takes M to a final state, we let $\mathcal{M}_C(M_\rho)$ denote the finite automaton with state set S that is equal to the set $Obs(M_\rho)$, alphabet $\mathcal{A}ct$, initial state (ϵ, ϵ) and the following set of transitions:*

- *Given $a \in I \cup \{\epsilon\}$, $a' \in I$ and a state $st = ((s_\alpha, a, s'_\alpha), b) \in S$, for all $st' = ((s'_\alpha, a', s''_\alpha), b) \in S$ there is a transition (st, a', st').*
- *Given $b \in O \cup \{\epsilon\}$, $b' \in O$ and a state $st = (a, (s_\alpha, b, s'_\alpha)) \in S$, for all $st' = (a, (s'_\alpha, b', s''_\alpha)) \in S$ there is a transition (st, b', st').*
- *Given $a \in I \cup \{\epsilon\}$, $a' \in I$ and a state $st = ((s_\alpha, a, s'_\alpha), b) \in S$, for all $st' = ((s_\beta, a', s'_\beta), b)$ there is a transition (st, a', st') if and only if there exists a path from s'_α to s_β labelled only with outputs.*
- *Given $b \in O \cup \{\epsilon\}$, $b' \in O$ and a state $st = (a, (s_\alpha, b, s'_\alpha)) \in S$, for all $st' = (a, (s_\beta, b', s'_\beta)) \in S$ there is a transition (st, b', st') if and only if there exists a path from s'_α to s_β labelled only with inputs.*

In addition, $\mathcal{M}_C(M_\rho)$ has a set of final states. A state $((s_\alpha, a, s'_\alpha), (s_\beta, b, s'_\beta)) \in S$ is final if there exists a walk $(s_0, a_0, s_1) \dots (s_{n-1}, a_{n-1}, s_n)$ with s_n a final state of M_ρ such that for some $0 \leq k, k' < n$ we have that $(s_k, a_k, s_{k+1}) = (s_\alpha, a, s'_\alpha)$, $(s_{k'}, a_{k'}, s_{k'+1}) = (s_\beta, b, s'_\beta)$ and for all $k < i < n$, $k' < j < n$, $(s_i, a_i, s_{i+1}) \notin I$ and $(s_j, a_j, s_{j+1}) \notin O$.

An important point is that we only consider a pair $st' = (a, b)$ if st' is in $Obs(M_\rho)$ and this requires that $a \bowtie_{M_\rho} b$. Thus, for example, the definition of $\bowtie_{M_\rho}$ ensures that we cannot have a state (a, b) if a is before b and they are separated only by inputs.

The final states are the states (a, b) such that a and b are the last input and output actions for some trace that reaches a final state of M_ρ.

Example 7. *Let us consider the finite automata M_ρ in Figure 9. The set of actions related under the observational relation for this automaton is*

$$\begin{aligned} Obs(M_\rho) \quad = \quad & \{(\epsilon, !x), (?a, \epsilon), (?b, \epsilon), (?c, \epsilon), (?d, \epsilon), (?e, \epsilon), (?g, \epsilon), \\ & (?a, !x), (?b, !x), (?c, !x), (?d, !x), (?e, !x), (?g, !x), \\ & (?a, !y), (?d, !y), (?e, !y), (?g, !y), (?g, !z), (?g, !w)\} \end{aligned}$$

Figure 9 also depicts the automaton $\mathcal{M}_C(M_\rho)$.

Proposition 15. *Let M be a Consistent FA and ρ a path of M that takes M to a final state. Given M_ρ with k transitions, the number of transitions of the automaton $\mathcal{M}_C(M_\rho)$ is of $O(k^3)$. In addition, the process of generating $\mathcal{M}_C(M_\rho)$ takes $O(k^3)$ time.*

Proof. We know, by Proposition 13, that the number of states in $\mathcal{M}_C(M_\rho)$ is of $O(k^2)$. Given a state s there are at most $k - 1$ transitions outgoing from s. Therefore, $\mathcal{M}_C(M_\rho)$ has $O(k^3)$ transitions.

Given M_ρ with k transitions, by Proposition 14, we can construct the set of states of $\mathcal{M}_C(M_\rho)$ in $O(k^2)$ time. In addition, in order to determine the transitions outgoing from a state (a, b) in $\mathcal{M}_C(M_\rho)$ it is sufficient to find each state (a', b') such that there exists a path in M_ρ between either a and a' or b and b' that satisfies the conditions imposed in Definition 11. This can be achieved in $O(k)$ time using two depth-first searches (one starting with a, the other starting with b). Therefore, the process of determining the transitions outgoing from a state (a, b) in $\mathcal{M}_C(M_\rho)$ takes time of $O(k)$. Since there are $O(k^2)$ states in $\mathcal{M}_C(M_\rho)$, the process of generating the transitions in $\mathcal{M}_C(M_\rho)$ is of $O(k^3)$ time. □

We now prove that $\mathcal{M}_C(M_\rho)$ is the finite automaton we require. In the proof of the following we use the notion of a transition x being *before* a transition x' where either x has an input label and x' has an output label or x has an output label and x' has an input label. This property (x is before x') simply refers to there being a path in M_ρ from the end state of x to the start state of x'. This defines a partial order on such pairs (x, x') of transitions since no cycle contains both inputs and outputs.

Proposition 16. *Given Consistent FA M and path ρ of M that takes M to a final state we have that $L(\mathcal{M}_C(M_\rho)) = \mathcal{L}(M_\rho)$.*

Proof. We will prove a slightly stronger result, which is that ρ' labels a walk from the initial state of $\mathcal{M}_C(M_\rho)$ if and only if ρ' is a prefix of a sequence in $\mathcal{L}(M_\rho)$.

We first prove the left to right implication by induction on the length of ρ'. The result clearly holds for the base case in which ρ' is the empty sequence. Now assume that the result holds for all sequences of length less than k, $k \geq 1$, and ρ' has length k. Thus, $\rho' = \rho_1 x$ for some $x \in I \cup O$. By the inductive hypothesis we have that ρ_1 is a prefix of a sequence in $\mathcal{L}(M_\rho)$. By the definition of $\mathcal{M}_C(M_\rho)$, we have that there exists a transition labelled with x from the state (a, b) reached by ρ_1 to a state (a', b') such that either $a' = x \wedge b' = b$ or $a' = a \wedge b' = x$. In the first case, we have that $x \in I$ and, by the construction of $\mathcal{M}_C(M_\rho)$, there exists a path in M_ρ from the final state of the transition a to the initial state of the transition $a' (= x)$ labelled only by outputs. Therefore, by the definition of $\mathcal{L}(M_\rho)$, we have that ρ_1 can be followed by x since it is possible that all the outputs between a and a' are delayed. In the second case, we have that $x \in O$ and, by the construction of $\mathcal{M}_C(M_\rho)$, there exists a path in M_ρ from b to b' labelled only with inputs. Since $(a, b') \in Obs(M_\rho)$ we have that, in M_ρ, either b' is before a (there is a path from b' to a) or b' is immediately after a (ab' is a path in M_ρ). In both situations, by the definition of $\mathcal{L}(M_\rho)$ we have that ρ_1 can be followed by x since the observation of b' can be delayed. This completes the proof by induction.

We now prove the right to left implication, again, by induction on the length of ρ'. The result clearly holds for the base case in which ρ' is the empty sequence. Now assume that the result holds for all sequences of length less than k, $k \geq 1$, and ρ' has length k. Thus, $\rho' = \rho_1 x$ for some $x \in I \cup O$. By the inductive

Algorithm 4 Producing $\mathcal{M}_C(A)$

1: Input M.
2: Let M' denote a finite automaton defining the empty language.
3: **for all** paths ρ of M that start in the initial state and end in a final state **do**
4: Construct M_ρ
5: Produce $\mathcal{M}_C(M_\rho)$
6: Update M' by taking the disjoint union with $\mathcal{M}_C(M_\rho)$ and then identifying their initial states.
7: **end for**
8: Output M'.

hypothesis we have that ρ_1 is the label of a walk of $\mathcal{M}_C(M_\rho)$ and so this walk reaches a state representing a pair (a, b). First consider the case where $x \in I$. By the definition of $\mathcal{L}(M_\rho)$ there is a path of M_ρ from the last input a in ρ_1 to x that contains only outputs and also we have that b is before x in M_ρ. Thus, $(x, b) \in ObsM_\rho$ and, by the definition of $\mathcal{M}_C(M_\rho)$, $\rho' = \rho_1 x$ is the label of a walk of $\mathcal{M}_C(M_\rho)$ as required. Now consider the case where $x \in O$. By the definition of $\mathcal{L}(M_\rho)$ there is a path of M_ρ from the last output b in ρ_1 to x that contains only inputs and also x is either before the last input a in ρ_1 or immediately follows a. As a result, $(a, x) \in ObsM_\rho$. Further, by the definition of $\mathcal{M}_C(M_\rho)$ we have that $\rho' = \rho_1 x$ is the label of a walk of $\mathcal{M}_C(M_\rho)$ as required. This completes the proof by induction.

□

Thus, for each path ρ to a final state we can obtain the automaton M_ρ from M. Then, we produce the finite automaton $\mathcal{M}_C(M_\rho)$ and define a finite automaton that accepts the union of the $\mathcal{M}_C(M_\rho)$. This approach is summarised in Algorithm 4. Since the approach takes as input an automaton it might seem that we could directly apply it to M but the following example shows that this need not work.

Example 8. *Let us consider the acyclic FA shown in Figure 6. In this case, for all* $1 \leq j, j' \leq k$, $j \neq j'$, *we have that* $(?i_j, !o_{j'})$, $(\epsilon, !o_j)$ *and* $(?i_j, \epsilon)$ *belong to* $Obs(M)$. *Then, by the construction of the automaton* $\mathcal{M}_C(M)$ *we will have that for all* $1 \leq k' < k'' \leq k$ *and for all* $1 \leq j \leq k$ *with* $j \neq k'$ *and* $j \neq k''$ *we have that* $((?i_{k'}, !o_j), ?i_{k''}, (?i_{k''}, !o_j))$, $((?i_j, !o_{k'}), !o_{k''}, (?i_j, !o_{k''}))$, $((?i_{k'}, \epsilon), ?i_{k''}, (?i_{k''}, \epsilon))$ *and* $((\epsilon, !o_{k'}), !o_{k''}, (\epsilon, !o_{k''}))$ *belong to the set of transitions of the automaton. Due to this fact, we have that for all* $1 \leq j < k$ *the traces* $?i_j i_{j+1} \ldots ?i_{k-1} ?i_k$ *and* $!o_j !o_{j+1} \ldots !o_{k-1} !o_k$ *are accepted by* $\mathcal{M}_C(M)$. *However, they do not belong to* $\mathcal{L}(M)$. *Thus, the set of traces accepted by* $\mathcal{M}_C(M)$ *do not correspond to the set of possible observations of the automaton* M.

This section has developed a method that deals with a class of cyclic automata. It would be interesting to look for more general conditions under which

$\mathcal{L}(M)$ is regular (and we can construct a finite automaton that recognises it) and maybe also conditions under which the process takes polynomial time.

6. Conclusions

This paper explored the situation in which we have a finite automaton M defining traces that the SUT should not have and the interactions between the SUT and the monitor are asynchronous and FIFO. Previous work showed how one can construct a finite automaton to be used by the monitor in the case where M is defined by a pair (ρ, O_ρ) that states that if the SUT produces trace ρ then the next output should be from O_ρ. The aim was to extend this approach to allow more general properties.

We first investigated an alternative approach, which is to operate on the trace σ observed rather than the finite automaton M that defines the property. We showed how one can construct a finite automaton $\mathcal{M}(\sigma)$ that defines the set of explanations for σ: the set of traces that the SUT might have produced given that σ has been observed. The problem then reduces to deciding whether the languages defined by M and $\mathcal{M}(\sigma)$ have a non-empty intersection. The main advantages of this approach are that it is general (it does not depend on the form of M) and also takes polynomial time. However, it has a potentially significant disadvantage: the size of $\mathcal{M}(\sigma)$ is of $O(|\sigma|^2)$ and so grows as more actions are observed. One would typically expect σ to become much larger than M and so one might prefer approaches that operate on M.

We then explored conditions under which the language $\mathcal{L}(M)$, of observations that might result from the SUT producing traces of M, is regular. First we constructed an approach that works if M is acyclic. In the worst case this can lead to a finite automaton M' whose size is exponential in terms of the size of M and we proved that this exponential growth cannot be avoided. We expect that, in practice, an exponential growth in the size of the (typically small) property M is preferable to a polynomial growth in the size of the (typically much larger) trace σ. However, this might not always be the case. We then generalised this approach to the case where every cycle of M has either only inputs or only outputs (but there might be cycles with inputs and others with outputs).

There are several possible lines of future work. First, there is the problem of finding weaker conditions under which we can generate a finite automaton to recognise $\mathcal{L}(M)$. It may also be possible to use a more general formalism than finite automata to represent $\mathcal{L}(M)$; we require a formalism for which the membership problem can be solved in low-order polynomial time. In practice, it may sometimes be possible to bound aspects of the problem such as the channel size and this might make the problem more tractable. We are working with the possibility of having probabilities associated with *swapping* the order of actions and reasoning in a probabilistic testing framework [28]. A further improvement would be to consider stochastic information regarding delays, that is, probability distributions, rather than fix bounds, would be used to represent fault models [15]. Finally, we are developing a prototype tool to support the

approach outlined in this paper and intend to carry out case studies. We will take as starting point our tool [8] dealing with our previous basic framework [16].

[1] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.

[2] C. Andrés, M. G. Merayo, and M. Núñez. Formal passive testing of timed systems: Theory and tools. *Software Testing, Verification and Reliability*, 22(6):365–405, 2012.

[3] J.A. Arnedo, A. Cavalli, and M. Núñez. Fast testing of critical properties through passive testing. In *15th Int. Conf. on Testing Communicating Systems, TestCom'03, LNCS 2644*, pages 295–310. Springer, 2003.

[4] A. Bauer. Monitorability of omega-regular languages. CoRR abs/1006.3638, 2010.

[5] A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *26th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'06, LNCS 4337*, pages 260–272. Springer, 2006.

[6] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4), 2011.

[7] E. Bayse, A. Cavalli, M. Núñez, and F. Zaïdi. A passive testing approach based on invariants: Application to the WAP. *Computer Networks*, 48(2):247–266, 2005.

[8] M. A. Camacho, M. G. Merayo, and I. Medina-Bulo. PTTAC: Passive Testing Tool for Asynchronous Systems. In *10th Int. Conf. on Signal-Image Technology & Internet-Based Systems, SITIS'14*, pages 223–229. IEEE Computer Society, 2014.

[9] A. R. Cavalli, T. Higashino, and M. Núñez. A survey on formal active and passive testing with applications to the cloud. *Annales of Telecommunications*, 70(3-4):85–93, 2015.

[10] X. Che and S. Maag. Passive performance testing of network protocols. *Computer Communications*, 51:36–47, 2014.

[11] M. Clerbout and M. Latteux. Semi-commutations. *Information and Computation*, 73(1):59–74, 1987.

[12] C. Colombo, G. J. Pace, and P. Abela. Safer asynchronous runtime monitoring using compensations. *Formal Methods in System Design*, 41(3):269–294, 2012.

[13] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[14] R. M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A.J.H Simons, S. Vilkomir, M.R. Woodward, and H. Zedan. Using formal specifications to support testing. *ACM Computing Surveys*, 41(2), 2009.

[15] R. M. Hierons, M. G. Merayo, and M. Núñez. Testing from a stochastic timed system with a fault model. *Journal of Logic and Algebraic Programming*, 78(2):98–115, 2009.

[16] R. M. Hierons, M. G. Merayo, and M. Núñez. Passive testing with asynchronous communications. In *IFIP 33rd Int. Conf. on Formal Techniques for Distributed Systems, FMOODS/FORTE'13, LNCS 7892*, pages 99–113. Springer, 2013.

[17] L. Ilie and S. Yu. Follow automata. *Information and Computation*, 186(1):140–162, 2003.

[18] O. Kupferman and M. Y. Vardi. Model checking of safety properties. In *11th Int. Conf. on Computer Aided Verification, CAV'99, LNCS 1633*, pages 172–183. Springer, 1999.

[19] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[20] T. Latvala. Efficient model checking of safety properties. In *10th Int. SPIN Workshop on Model Checking of Software, SPIN'03, LNCS 2648*, pages 74–88. Springer, 2003.

[21] D. Lee, D. Chen, R. Hao, R.E. Miller, J. Wu, and X. Yin. Network protocol system monitoring: a formal approach with passive testing. *IEEE/ACM Transactions on Networking*, 14:424–437, 2006.

[22] D. Lee, A.N. Netravali, K.K. Sabnani, B. Sugla, and A. John. Passive testing and applications to network management. In *5th IEEE Int. Conf. on Network Protocols, ICNP'97*, pages 113–122. IEEE Computer Society, 1997.

[23] M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[24] A. Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency, LNCS 255*, pages 278–324. Springer, 1987.

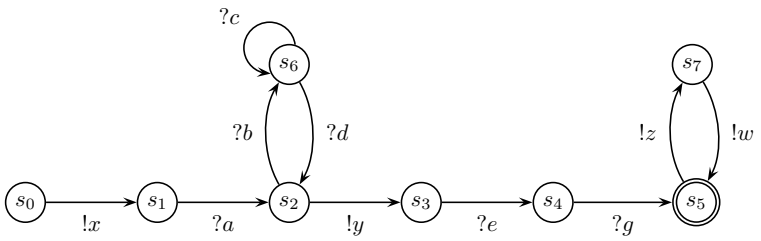
[25] M. G. Merayo and A. Núñez. Passive testing of communicating systems with timeouts. *Information and Software Technology*, 64:19–35, 2015.

[26] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, 2nd edition, 2004.

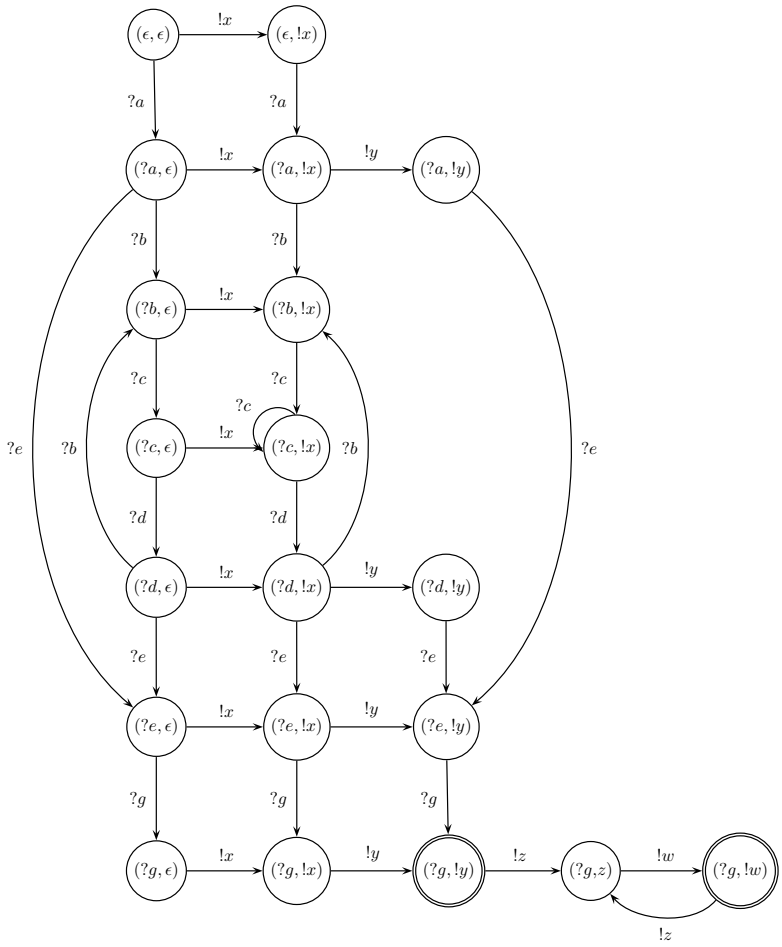
[27] H. N. Nguyen, P. Poizat, and F. Zaïdi. Online verification of value-passing choreographies through property-oriented passive testing. In *14th Int. Symposium on High-Assurance Systems Engineering, HASE'12*, pages 106–113. IEEE Computer Society, 2012.

[28] M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.

[29] R.E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.



(a) M_ρ



(b) $\mathcal{M}_C(M_\rho)$

Figure 9: Finite automata M_ρ and $\mathcal{M}_C(M_\rho)$