Distributed Computing manuscript No.
(will be inserted by the editor)

Passive Testing with Asynchronous Communications and Timestamps

Mercedes G. Merayo, Robert M. Hierons and M. Núñez

July 25th, 2017

Abstract We develop a formal passive testing framework for software systems where parties communicate asynchronously. Monitors, placed in between the entities, check that a certain property holds over the observations of the interaction between users and the System Under Test (SUT). Due to the asynchronous nature of communications, the trace observed by the monitor might differ from the one produced by the SUT: the monitor observes inputs before they are received by the SUT and outputs are observed after they are sent by the SUT. It is necessary to take this into account in passive testing; otherwise we might obtain false positives or false negatives. In order to better assess the real causality between actions, we consider the case where each action is labelled with a timestamp giving the time when it was observed at the monitor. We also assume that we know bounds on network latency and so the timestamps allow us to determine additional causalities between actions. Our monitors are implemented as automata that take into account communications being asynchronous. Our solution checks properties against traces in polynomial time and has low storage requirements. Therefore, our proposal is suitable for real-time passive testing.

Keywords Model-based testing; Distributed systems; Timed systems

Research partially supported by the projects DArDOS (TIN2015-65845-C3-1-R (MINECO/FEDER)) and SICOMORo-CM (S2013/ICE-3006).

M. G. Merayo and M. Núñez
Departamento de Sistemas Informáticos y Computación,
Universidad Complutense de Madrid, Spain
E-mail: mgmerayo@fdi.ucm.es , mn@sip.ucm.es

R. M. Hierons
Department of Information Systems and Computing,
Brunel University, Uxbridge, Middlesex, UB8 3PH United Kingdom
E-mail: Rob.Hierons@brunel.ac.uk

1 Introduction

Current software systems are very complex and consist of many heterogeneous components. Therefore, ensuring that the developed systems are correct, that is, they respect the requirements defined by the designer of the system, is an extremely arduous task. Thus, it is important to provide sound engineering techniques to increase the confidence regarding the correctness of these systems. Software testing is one of the most widespread validation techniques [44]. The main problem with testing is that it is mainly a manual activity, increasing the cost of the system. In fact, it is recognised that the cost of testing can be more than 50% of the total budget [44]. Therefore, the development of automated testing techniques is a must and one way of achieving this is to have a formal description of the system that we are testing and of the properties that we would like to test. While the testing of software has traditionally had a weak formal basis, testing of hardware systems originally included formalisms and formal notations [38,15]. Only relatively recently, has it been recognised that *formalising* the different aspects of testing software is very beneficial [12]. Currently, a formal discipline of software testing, combining formal methods and testing, is well understood, tools to automate testing activities are widely available [37], and there are several surveys of the field [20,19,8]. In addition, industry is becoming aware of the importance of using formal approaches to testing in different application areas [31,13].

The most common understanding of testing is that it is a process in which a tester applies inputs to the SUT, observes the resultant outputs, and makes a verdict on whether the outputs were correct. However, often there is a need to assess the correctness of a system without having direct access to this system. This could happen due to security issues or because the system is running 24/7 and an interaction with it might produce undesirable changes in the associated data. In these situations, testing can still play a role if we consider it to be a *monitoring* activity where interaction is replaced by observation. This approach to testing is also known as *passive testing* and it is already a well established line of research [34,7,4,39] where extensions of the original frameworks have gone beyond the classical application to validate protocols [32,33,45,43,30,9] to deal with issues such as security [48,42,36,1]. Essentially, in passive testing we have a property and we check that the trace (sequence of inputs and outputs) being observed satisfies the property. Ideally, the process of checking whether the property is satisfied should be quick and take very little storage since this can allow passive testing to occur in real-time. In fact, the application of passive testing in real-time has an important benefit: the operators of the system can be notified of a detected error almost immediately and they can then take appropriate measures. If the trace has to be saved and processed off-line, then the time between the detection of the error and the corresponding notification will significantly increase.

Previous work on passive testing has assumed that the monitor observes the actual trace produced by the SUT. However, the monitor might not directly observe the interface of the SUT and instead there may be an asynchronous channel/network between the monitor and the SUT. The analysis of systems in an asynchronous setting requires us to consider not only the traces that can be performed by the system but also how these traces can be observed. Due to the existence of an asynchronous channel/network between the monitor and the SUT, the trace observed by the monitor might not be the one actually produced by the

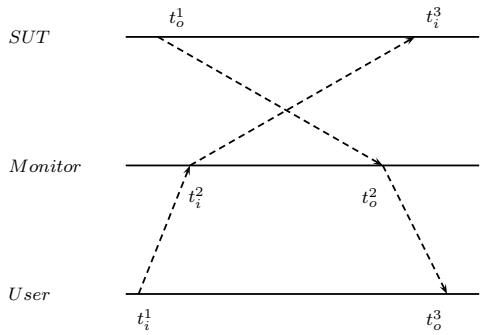


Fig. 1 Actions swapped in the monitor w.r.t. the system

SUT. The main problem is that input is observed before it is received by the SUT and output is observed after it is sent by the SUT.

In Figure 1 output o is observed by the monitor after it is produced ($t_o^1 < t_o^2$) and the input i is observed before it is received by the SUT ($t_i^2 < t_i^3$). In addition, it happens that the output o is produced before the input i is received ($t_o^1 < t_i^3$), but the observation is different ($t_i^2 < t_o^2$). Therefore, the sequence of actions observed at the monitor might contain *swaps* with respect to the one that was produced at the SUT.

Thus, if we have properties that the SUT should satisfy and we directly apply them in such a context then we may obtain false positives or false negatives. We therefore require new approaches to passive testing in such circumstances and in this paper we focus on the case where the asynchronous channels are first in first out (FIFO).

In our previous work [22] we presented a passive testing framework where the monitor observes sequences of actions. We have also developed a tool that implements the proposed framework. This tool has been used to evaluate different properties of communication protocols [6]. In this paper we consider the case where observed actions are stamped with the time when they were observed by the monitor. We analyse how this information can be used to reason about the precise order in which events were produced. For example, let us consider a property that states that if a system produces the sequence of actions $?i!o$ for input $?i$ and output $!o$, then the next output must be from a set O'. If the monitor observes $?i$ followed by $!o$ then it is not possible to claim that this was the order in which the actions were produced: they might have been produced in the reverse order with the observation being due to a delay. However, if we are also provided with timestamps then we might be able to more accurately establish the actual order in which the events were produced by the SUT. For example, let us suppose that we want to detect sequences of actions in which the output $!o$ is produced before the input $?i$ is received. If we observe $(!o, 100), (?i, 101)$ then we know that $!o$ was performed before $?i$. However, if the monitor observes $(?i, 100), (!o, 101)$ then we cannot claim that $!o$ was produced after the reception of $?i$ since all we know is that the output was produced before 101 and the input was received after 100. In an asynchronous setting we must take into account message latency when we reason about the order in which actions occurred. In the previous case, if we have a lower bound on message latency of 0.6, then we know that $?i$ was received by

the SUT not before 100.6 while $!o$ was performed not later than 100.4. Therefore, we can conclude that $?i$ was received by the SUT after $!o$ was emitted.

In this paper we present a complete formal framework to perform passive testing of software systems with asynchronous communications where actions are timestamped with the time when they were observed at the monitor. Essentially, a monitor is implemented as an automaton that checks, with a low storage overhead and with operations taking place in low-order polynomial time, whether the sequence of observed actions might fulfill a property such as "if we observe a sequence of actions then the next observed output belongs to a certain set". Actually, we will prove that the time complexity of these operations is in $O(n^4)$, where n is the length of the property. Since the usual properties that we foresee are very short, it is indeed feasible to use our approach in real-time.

We build on top of previous work [22]. Even though we are using the same type of properties, the inclusion of time complicates the definitions of most of the concepts and a new, more complex, algorithm is needed to generate automata from properties.

We are not aware of other work on passive testing where there is an asynchronous communications channel between the system and the monitor. In contrast, there has been some work on active testing for models with asynchronous communications where there is a distinction between inputs and outputs [16,27,49, 17,18,47]. Concerning the consideration of time, there is plenty of work on formal approaches to both active [50,41,21,29,40,23,46] and passive [42,3] testing of timed systems. However, as far as we know, these approaches assume a synchronous communications mechanism. In addition, we use time, as simple timestamps, to gather more information about the causality between actions while in these approaches time is used to represent requirements and properties of the analysed systems. There exists an approach using timestamps to generate tests in asynchronous systems [28]. Timestamps are added by the SUT and the SUT has to log the events with timestamps. Thus, the SUT must be modified, and this is a drawback, but the main advantage is that one can reconstruct the trace that the SUT performed. Our approach does not require the SUT to be changed and does not timestamp the events at the SUT; it instead requires that the monitor adds timestamps. Passive testing is a monitoring technique and as such it is related to runtime verification [35] since they share the same goal, checking the correctness of a system without interacting with it, but use different formalisms and methodologies. In runtime verification it is not usual to distinguish between inputs and outputs, since their observation makes them events of the same nature, and therefore it is difficult to compare the work from that area with ours. While some work has investigated asynchronous runtime monitoring, the problems considered in this context are different: this line of work does not distinguish between input and output and does not explore potential reorderings of traces. Instead, it looks at the situation in which the monitor and the system do not synchronise on actions: actions engaged in by the system might instead be recorded and analysed later, with a compensation phase being used to undo any later actions if an error is found [10].

The rest of the paper is structured as follows. In Section 2 we introduce notation to define systems and traces that will be used throughout the paper. We also discuss different alternatives to perform passive testing in systems with asynchronous communications and advance our solution. Section 3 introduces the main

notions of the paper, in particular, how traces are annotated with time. Section 4 is the bulk of the paper and presents how properties can be translated into automata and provides our theory to check traces against properties. Finally, in Section 5 we present our conclusions and provide some lines for future work.

2 Preliminaries

In this section we introduce the basic notation used in this paper to define systems as well as concepts associated with the traces that a system can perform and with the traces that a monitor can actually observe in an asynchronous setting. We also review alternative options to implement our approach, present an impossibility result and outline our solution. The material presented in this section is taken from our previous work [22].

Definition 1 An *input-output transition system (IOTS)* $M = (Q, I, O, T, q_{in})$ is a tuple in which Q is a countable set of states, $q_{in} \in Q$ is the initial state, I is a countable set of inputs, O is a countable set of outputs, and $T \subseteq Q \times (I \cup O) \times Q$ is the transition relation. A transition $(q, a, q') \in T$ means that from state q it is possible to move to state q' with action $a \in I \cup O$.

We use the following notation concerning the performance of (sequences of) actions.

- $\mathcal{A}ct = I \cup O$ is the set of actions.
- If $(q, a, q') \in T$, for $a \in \mathcal{A}ct$, then we write $q \xrightarrow{a} q'$ and $q \xrightarrow{a}$.
- We write $q \stackrel{\sigma}{\Longrightarrow} q'$ for $\sigma = a_1 \dots a_m \in \mathcal{A}ct^*$, with $m \geq 0$, if there exist $q_0, \dots, q_m$, $q = q_0$, $q' = q_m$ such that for all $0 \leq i < m$ we have that $q_i \xrightarrow{a_{i+1}} q_{i+1}$. Note that $q \stackrel{\epsilon}{\Longrightarrow} q$, where ϵ is the empty sequence.
- If there exists q' such that $q_{in} \stackrel{\sigma}{\Longrightarrow} q'$ then we say that σ is a *trace* of M and we write $M \stackrel{\sigma}{\Longrightarrow}$. We let $L(M)$ denote the set of traces of M.

During the rest of the paper when we refer to a system we will assume that there is an IOTS representing the behaviour of that system and, in such a context, we will use both terms as interchangeable. In addition, we will fix the sets of inputs, outputs and the total set of actions to be I, O and $\mathcal{A}ct$, respectively. In order to distinguish between input and output we usually precede the name of an input by ? and precede the name of an output by !.

We have an asynchronous setting and, therefore, we do not only have to consider the traces that can be performed by a system but also how these traces can be observed. Intuitively, 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. 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. Next we formally define this idea and given a system M and a trace σ we let $\mathcal{L}(\sigma)$ denote the set of traces that might be observed by a monitor if M produces trace σ and communications between the monitor and the SUT are asynchronous and FIFO.

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 zero or more transformations of the form $\rightsquigarrow$, that is, $\mathcal{L}(\sigma) = \{\sigma' | \sigma \rightsquigarrow^* \sigma'\}$, where $\rightsquigarrow^*$ represents the repeated application of $\rightsquigarrow$ zero or more times. We overload this to say that given an IOTS 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.

Example 1 Let us assume that the SUT has produced the trace $\sigma = ?i_1 !o_1 !o_2 ?i_2 !o_1$. Due to the asynchronous nature of communications, the monitor might observe any of the traces in the set $\mathcal{L}(\sigma) = \{?i_1 !o_1 !o_2 ?i_2 !o_1, ?i_1 !o_1 ?i_2 !o_2 !o_1, ?i_1 ?i_2 !o_1 !o_2 !o_1\}$.

In line with previous work in formal passive testing [4], we will consider properties of the form (σ, O_σ) for $\sigma \in \mathcal{A}ct^*$ and $O_\sigma \subseteq O$. Such a property says that if the SUT produces the sequence σ then the next output must come from the set O_σ. One possible approach to working with properties and traces is to represent the property P as an automaton $M(P)$ such that a trace satisfies P if and only if it is not a member of the language defined by $M(P)$: it does not reach a final (error) state of $M(P)$, that is, $M(P)$ accepts the regular language $\mathcal{A}ct^* \{\sigma\} (\mathcal{A}ct \setminus O_\sigma) \mathcal{A}ct^*$. If $M(P)$ is deterministic then the process of checking whether a trace satisfies P takes linear time and is an incremental process: every time we observe a new input or output we simply update the state of $M(P)$. Even if $M(P)$ is non-deterministic, the process of checking whether a trace satisfies P takes time that is quadratic in the size of $M(P)$ and linear in the length of the trace and the process is still incremental. This makes such an automaton based approach desirable if we can find efficient ways of mapping properties to automata. It is well known that an automaton that represents a regular expression can be produced in quadratic time [5]. This process can be further improved to achieve sub-quadratic complexity and can be efficiently parallelised to work in $O(\log(|\sigma|))$ time [14]. Such an automaton $M(P)$ has $O(|\sigma|)$ states and $O(|\sigma| \cdot \log(|\sigma|)^2)$ transitions [25]. In the next section we adapt the above approach for the case where communications are asynchronous and actions are stamped with the time when they were observed at the monitor.

There are several ways of applying passive testing when observations are through FIFO channels. Before outlining our solution, we briefly comment on some alternatives. One approach creates a model of the property and adds queues to this. However, the addition of queues can lead to the model requiring more storage space. In particular, if the queues are not bounded then it leads to there being an infinite number of states while if there is a bound on the queue length then the number of states increases exponentially with this bound. An alternative is to transform the trace ρ observed to form an automaton M_ρ that represents all traces that might lead to ρ being observed where there is asynchronous communications. However, under this approach we have to analyse the entire, potentially very long, trace that has been observed and the monitor has to store this. This approach thus mitigates against real-time uses since it can significantly increase the storage and processing requirements. Previous work has described a delay operator that takes a trace σ of an IOTS and returns the set of traces that might be observed if the SUT produces σ and interacts asynchronously through FIFO channels with its environment [26]. However, the delay operator cannot be applied directly since it applies to a single trace rather than an automaton $M(P)$. While it has been shown

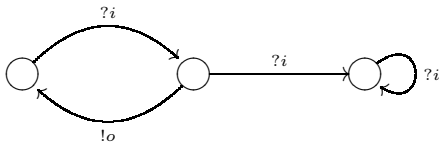


Fig. 2 An IOTS M such that $\mathcal{L}(M)$ is not regular.

how a test purpose can be adapted to incorporate the delay into the verdict [49], this approach assumes that the tester waits for output before sending the next input and so does not apply input in a state where output can be produced. Since the input is not supplied by the tester in passive testing, we cannot make such an assumption. Finally, we might instead aim to define a general method that takes an IOTS M (with finite sets of states and transitions) and produces IOTS M' (with finite sets of states and transitions) with $L(M') = \mathcal{L}(M)$. If we can achieve this then M' can be used. However, the following shows that there is no such general method.

Proposition 1 *Given an IOTS M with finite sets of states and transitions, there may be no IOTS M' with finite sets of states and transitions such that $L(M') = \mathcal{L}(M)$.*

Proof An IOTS with finite sets of states and transitions defines a regular language so it is sufficient to find some such M where $\mathcal{L}(M)$ is not a regular language. Let M be the IOTS with three states shown in Figure 2 (the initial state is represented by the leftmost vertex). We will use proof by contradiction, assuming that $\mathcal{L}(M)$ is a regular language. Thus, since $\mathcal{L}(M)$ is regular and I^*O^* is regular we have that $\mathcal{L}(M) \cap (I^*O^*)$ is regular. However, $\mathcal{L}(M) \cap (I^*O^*)$ contains all sequences of the form of n inputs followed by n or fewer outputs and this is not a regular language. This provides a contradiction as required.

While this result shows that there is no general method that takes a property P defined by an IOTS $M(P)$ with finite sets of states and transitions and returns a suitable property for use when communications are asynchronous, we will see that we can take advantage of the structure of the properties we consider and the timestamps associated to the observation of the trace. First we will show how, for trace σ, we can produce an automaton that gives the set of traces that might be observed if the SUT produces σ. This automaton will include conditions over the timestamps that allow us to check whether the causality relation between the actions from the observed trace might correspond to the production of the expected trace. Since we are applying passive testing, a trace σ of interest might not be the start of the overall trace observed and so we will then adapt the previous automaton to produce the automaton that will be used.

3 Adding timestamps to observations

In this section we introduce the main notions of our framework to include the time information compiled while observing actions at the monitors.

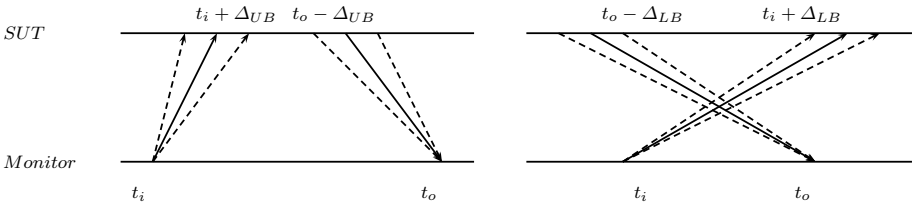


Fig. 3 An input before an output (left) and an output before an input (right).

Definition 3 We assume that the time domain includes all non-negative real numbers, that is, Time $= \mathbb{R}_+$. Given $(a,t) \in \mathcal{A}ct \times$ Time we have that $\text{act}(a,t) = a$ and $\text{time}(a,t) = t$. Let $\sigma \in (\mathcal{A}ct \times \text{Time})^*$ be a sequence of (action, time) pairs. For all $1 \leq i \leq |\sigma|$ we denote by σ_i the i-th element of the trace.

We let $\text{untime}(\sigma)$ denote the trace produced from σ by removing the timestamps associated with actions. Formally,

$$\text{untime}(\sigma) = \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ a\ \text{untime}(\sigma') & \text{if } \sigma = (a,t)\sigma' \end{cases}$$

A *timed trace* is a sequence $\sigma \in (\mathcal{A}ct \times \text{Time})^*$ such that for all $i < j$ we have $\text{time}(\sigma_i) < \text{time}(\sigma_j)$.

Timed traces are sequences of inputs and outputs annotated with the time at which actions were observed. The timestamps define the exact order in which actions were observed. We assume that two actions cannot be timestamped with the same value since it takes time to observe an action.

In order to reason about the causality between actions that have been observed in an asynchronous setting, taking into account that these observations are timestamped, it is necessary to make assumptions regarding message latency, that is, the amount of time needed for a message to be transmitted from the SUT/monitor to the monitor/SUT through the network. We assume that there is a known lower bound Δ_{LB} on message latency and also a known upper bound Δ_{UB}. We also assume that we have the same bound in each direction; it is straightforward to adapt the definitions and results to the case where bounds differ. We used similar assumptions and notation in our previous work on testing in the distributed architecture [23].

Since communications are FIFO, we can always order two inputs and also two outputs, once they have been observed. We must consider the situation in which we have an input and an output. If the output is observed before the input then clearly the output was produced by the SUT before the input was received. It is therefore sufficient to consider the situation in which an input $?i$ is observed at time t_i before an output $!o$ is observed at time t_o. Then, $?i$ must have been received by the SUT before time $t_i + \Delta_{UB}$ and $!o$ must have been sent by the SUT after time $t_o - \Delta_{UB}$. Thus, we know that $!o$ was sent after $?i$ was received if $t_i + \Delta_{UB} < t_o - \Delta_{UB}$ and this is the case if and only if $t_o - t_i > 2\Delta_{UB}$. Similarly, we know that $!o$ was sent before $?i$ was received if $t_o - \Delta_{LB} < t_i + \Delta_{LB}$ and this is the case if and only if $t_o - t_i < 2\Delta_{LB}$. This generalises the situation in which

an output is observed before an input, showing that we can potentially know that an output was produced before an input was received if the output is observed less than $2\Delta_{LB}$ time units after the input was sent. These situations are shown in Figure 3 in which time progresses as we move to the right, a solid arc represents the message (an input or an output), and the dotted arcs represent the bounds on when the message might have been received/sent.

We need to define the set of timestamped traces that might be observed by the monitor if the SUT produces a trace in an asynchronous FIFO setting where the bounds on message latency are known. Essentially, we refine our previous relation $\rightsquigarrow^*$ to take into account time information. A sequence can present multiple occurrences of a specific action. In order to distinguish the different occurrences of the same action in the sequence we use events. An event is a pair that associates to each action a number that indicates the occurrence of the action in the sequence. For example, the event $(a, 1)$ corresponds to the first occurrence of an action a in the sequence, the event $(a, 2)$ corresponds to the second occurrence of a in the sequence and so on. We now formalise this approach.

We transform traces into sets of events. Given a trace σ, we derive a set of events that allows us to distinguish between repeated actions in the trace. The elements are constructed from actions by labelling each action in σ with the occurrence of the symbol in the trace. For example, in the trace $?i_1!o_1?i_1$ there are two instances of the action $?i_1$: the event (pair) $(?i_1, 1)$ (denoting the first occurrence of $?i_1$ in the trace) and the event $(?i_1, 2)$ (denoting the second occurrence of $?i_1$ in the trace). We will also say that each such event has a position in the original trace. For example, the position of $(?i_1, 1)$ in trace $?i_1!o_1?i_1$ is 1 since the event $(?i_1, 1)$ (the first occurrence of $?i_1$) corresponds to the first action in $?i_1!o_1?i_1$. Similarly, the position of $(?i_1, 2)$ in trace $?i_1!o_1?i_1$ is 3 since the event $(?i_1, 2)$ (the second occurrence of $?i_1$) corresponds to the third action in $?i_1!o_1?i_1$.

Definition 4 Let $\sigma = a_1 \dots a_n \in \mathcal{A}ct^*$ be a sequence of actions. We let $E(\sigma)$ denote the *set of events of* σ, where $e = (a_i, k)$ belongs to $E(\sigma)$ if and only if there are exactly $k-1$ occurrences of a_i in $a_1 \dots a_{i-1}$. We define the function $pos_\sigma : E(\sigma) \longrightarrow \mathbf{N}$ 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 define the *annotated sequence of* σ, denoted by $\widetilde{\sigma}$, as the sequence $(a_1, k_1) \dots (a_n, k_n)$. Respectively, the *annotated timed sequence of* a timed trace $\sigma_t = (a_1, t_1) \dots (a_n, t_n) \in (\mathcal{A}ct \times \text{Time})^*$ is defined as the sequence $((a_1, k_1), t_1) \dots ((a_n, k_n), t_n)$.

Usually, we will decorate an event with its position. For example, e_i denotes that $pos_\sigma(e) = i$.

A sequence can present multiple occurrences of a specific action. In order to distinguish the different occurrences of the same action in the sequence we use events. An event is a pair that associates to each action a number that indicates the occurrence of the action in the sequence. For example, the event $(a, 1)$ corresponds to the first occurrence of an action a in the sequence, the event $(a, 2)$ corresponds to the second occurrence of a in the sequence and so on.

Example 2 Consider the trace $\sigma = ?i_1!o_1?i_2!o_1!o_2$. The corresponding set of events is

$$E(\sigma) = \{(?i_1, 1), (?i_2, 1), (!o_1, 2), (!o_1, 1), (!o_2, 1)\}$$

where, for example, $pos_\sigma((?i_1, 1)) = 1$ and $pos_\sigma((!o_1, 1)) = 2$. We will refer to event $(?i_2, 1)$ by e_3.

We now define the set of possible timed traces, given Σ, if the SUT produces untimed trace σ. For this we require two constraints. First, if σ_t is a timed trace that resulted from the SUT producing trace σ then we require that the observation of the untimed trace $untimed(\sigma_t)$ is consistent with FIFO communications ($\sigma \rightsquigarrow^*$ untime(σ_t)). Second, we note that Δ sometimes allows us to determine, for two timed events $((a_i, k_i), t_i)$ and $((a_j, k_j), t_j)$ in σ_t, the relative order of the corresponding actions of the SUT; this must be consistent with σ.

Definition 5 Let $\sigma \in Act^*$ be a trace, $\sigma_t \in (Act \times \text{Time})^*$ be a timed trace, and $\Delta = (\Delta_{LB}, \Delta_{UB}) \in \text{Time} \times \text{Time}$. We say that σ_t is a timed observation of σ for Δ, denoted by $\sigma \rightsquigarrow_\Delta \sigma_t$, if $\sigma \rightsquigarrow^*$ untime(σ_t) and for all $!o \in O$, $?i \in I$, $\sigma_1, \sigma_2 \in Act^*$ we have that

- if $\widetilde{\sigma} = \widetilde{\sigma}_1(!o, k_o)(?i, k_i)\widetilde{\sigma}_2$ then there exist $\sigma'_1, \sigma'_2, \sigma'_3 \in (Act \times \text{Time})^*$ such that either
  - $\widetilde{\sigma}_t = \widetilde{\sigma}'_1((!o, k_o), t_o)((?i, k_i), t_i)\widetilde{\sigma}'_2$ or
  - $\widetilde{\sigma}_t = \widetilde{\sigma}'_1((?i, k_i), t_i)\widetilde{\sigma}'_2((!o, k_o), t_o)\widetilde{\sigma}'_3 \wedge t_o - t_i \leq 2\Delta_{UB}$
- if $\widetilde{\sigma} = \widetilde{\sigma}_1(?i, k_i)(!o, k_o)\widetilde{\sigma}_2$ then there exist $\sigma'_1, \sigma'_2, \sigma'_3 \in (Act \times \text{Time})^*$ such that

$$\widetilde{\sigma}_t = \widetilde{\sigma}'_1((?i, k_i), t_i)\widetilde{\sigma}_2((!o, k_o), t_o)\widetilde{\sigma}_3 \wedge t_o - t_i \geq 2\Delta_{LB}$$

We let $\mathcal{L}_\Delta(\sigma)$ denote the set of timed observations of σ for $\Delta = (\Delta_{LB}, \Delta_{UB})$, that is, $\mathcal{L}_\Delta(\sigma) = \{\sigma' | \sigma \rightsquigarrow_\Delta \sigma'\}$. We overload this to say that given a system M, $\mathcal{L}_\Delta(M) = \cup_{\sigma \in L(M)} \mathcal{L}_\Delta(\sigma)$ is the set of timestamped traces that might be observed when interacting with M through asynchronous FIFO channels with Δ_{LB} and Δ_{UB} as the lower and upper bounds on message latency, respectively.

Let us note that the conditions imposed by the definition of a timed observation might be applied several times for each trace.

Example 3 Let us consider the following two sequences: $\sigma = !o_1?i_1!o_2?i_2!o_3$ and $\sigma_t = (?i_1, t_1)(!o_1, t_2)(?i_2, t_3)(!o_2, t_4)(!o_3, t_5)$. In order to determine whether σ_t is a timed observation of σ, we need to perform different evaluations of the conditions, taking into account the following cases:

- $\sigma = \epsilon \cdot !\mathbf{o_1}?\mathbf{i_1} \cdot !o_2?i_2!o_3$ and
  $\sigma_t = \epsilon \cdot (?\mathbf{i_1}, \mathbf{t_1}) \cdot \epsilon \cdot (!\mathbf{o_1}, \mathbf{t_2}) \cdot (?i_2, t_3)(!o_2, t_4)(!o_3, t_5)$ then $t_2 - t_1 \leq 2\Delta_{UB}$
- $\sigma = !o_1?i_1 \cdot !\mathbf{o_2}?\mathbf{i_2} \cdot !o_3$ and
  $\sigma_t = (?i_1, t_1)(!o_1, t_2) \cdot (?\mathbf{i_2}, \mathbf{t_3}) \cdot \epsilon \cdot (!\mathbf{o_2}, \mathbf{t_4}) \cdot (!o_3, t_5)$ then $t_4 - t_3 \leq 2\Delta_{UB}$
- $\sigma = !o_1 \cdot ?\mathbf{i_1}!\mathbf{o_2} \cdot ?i_2!o_3$ and
  $\sigma_t = \epsilon \cdot (?\mathbf{i_1}, \mathbf{t_1}) \cdot (!o_1, t_2)(?i_2, t_3) \cdot (!\mathbf{o_2}, \mathbf{t_4}) \cdot (!o_3, t_5)$ then $t_4 - t_1 \geq 2\Delta_{LB}$
- $\sigma = !o_1?i_1!o_2 \cdot ?\mathbf{i_2}!\mathbf{o_3} \cdot \epsilon$ and
  $\sigma_t = (?i_1, t_1)(!o_1, t_2) \cdot (?\mathbf{i_2}, \mathbf{t_3}) \cdot (!o_2, t_4) \cdot (!\mathbf{o_3}, \mathbf{t_5}) \cdot \epsilon$ then $t_5 - t_3 \geq 2\Delta_{LB}$

In particular, we might have the conjunction of several temporal conditions associated with the same timed trace.

Example 4 Let us suppose that the SUT has produced the trace $\sigma = ?i_1!o_1?i_2$ and the lower and upper bounds on message latency are 0.1 and 0.15 respectively. The monitor might observe any of the timed traces belonging to the set

$$\begin{aligned}\mathcal{L}_{(0.1,0.15)}(\sigma) = \; & \{(?i_1,t_1)(!o_1,t_2)(?i_2,t_3) | t_2 - t_1 \geq 0.2\} \bigcup \\ & \{(?i_1,t_1)(?i_2,t_2)(!o_1,t_3) | t_3 - t_1 \geq 0.2 \wedge t_3 - t_2 \leq 0.3\}\end{aligned}$$

In the rest of this section we introduce the crucial notion of *ideal* and present some simple properties. This structure will be useful when defining the states of the automata representing the properties to be checked by the monitors.

Given a sequence σ, we will define a partial order $\ll$ on the inputs and outputs in σ to represent which actions must be observed before others if the SUT produces σ. Thus, if $\sigma = a_1 \ldots a_n$, $i < j$, $e_i = (a_i, k_i)$, and $e_j = (a_j, k_j)$ then we will use $e_i \ll e_j$ to denote us knowing that e_i must be observed before e_j. Since outputs are delayed and inputs are not delayed, this relation must hold if the second event is an output and the first is an input. Further, since communications are FIFO, the relation must also hold if a_i and a_j are both inputs or a_i and a_j are both outputs; one input cannot overtake another and one output cannot overtake another.

Definition 6 Let $\sigma = a_1 \ldots a_n \in \mathcal{A}ct^*$ be a sequence of actions. Given two events $e_i = (a_i, k_i)$ 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 first two cases in the definition of $\ll$ result from channels being FIFO. The last case results from the observation of outputs being delayed, while an input is observed before it is received by the SUT. Essentially, $(a_i, k_i) \ll (a_j, k_j)$ does not hold for $i < j$ if a_i is an output and a_j is an input since in this case it is possible that the observation of output a_i is delayed until after input a_j has been sent.

Example 5 Let us consider again the trace $\sigma = ?i_1!o_1?i_2!o_1!o_2$. For instance, we have that $(?i_1, 1) \ll (!o_1, 2)$ while $(!o_1, 1) \ll (?i_2, 1)$ does not hold.

Given a trace $\sigma \in \mathcal{A}ct^*$ it is straightforward to prove that $(E(\sigma), \ll)$ is a partially ordered set. Next we recall the definition of an *ideal*.

Definition 7 Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $E(\sigma)$ be the set of its events. A set $\mathcal{I} \subseteq E(\sigma)$ is said to be an *ideal* of $(E(\sigma), \ll)$ if for all $e_i, e_j \in E(\sigma)$, if $e_i \ll e_j$ and $e_j \in \mathcal{I}$ then $e_i \in \mathcal{I}$. An ideal $\mathcal{I}$ is a *principal ideal* if there is some e_j such that $\mathcal{I}$ contains only e_j and all elements below it under $\ll$, that is, $\mathcal{I} = \{e_i \in E(\sigma) | e_i \ll e_j\}$. Finally, a set $E' \subseteq E(\sigma)$ is an *anti-chain* if no two different elements of E' are related under $\ll$.

Intuitively, if the SUT produces σ and e_i is a maximal element of ideal $\mathcal{I}$, then $\mathcal{I}$ includes all events that *must* be observed before e_i is observed by the monitor.

Example 6 Consider the trace $\sigma = ?i_1!o_1?i_2!o_1!o_2$. The following sets of events are ideals of $(E(\sigma), \ll)$.

$$\begin{aligned}\mathcal{I}_1 &= \{(?i_1, 1), (?i_2, 1)\} \\ \mathcal{I}_2 &= \{(?i_1, 1), (!o_1, 1)\} \\ \mathcal{I}_3 &= \{(?i_1, 1), (!o_1, 1), (?i_2, 1)\}\end{aligned}$$

However, only $\mathcal{I}_1$ and $\mathcal{I}_2$ are principal ideals. The ideal $\mathcal{I}_3$ contains the events $(!o_1, 1)$ and $(?i_2, 1)$ that are not related under $\ll$. In addition, there are no elements of $\mathcal{I}_3$ that are 'above' $(!o_1, 1)$ and $(?i_2, 1)$ under $\ll$. As a result, $\mathcal{I}_3$ is not a principal ideal.

Now consider the set $E_1 = \{(!o_1, 1), (?i_2, 1)\}$. The two events are unrelated by $\ll$ and so this is an anti-chain of $(E(\sigma), \ll)$.

Next we present an alternative characterisation of the notion of an ideal that shows that an ideal is defined by its maximal (under $\ll$) elements. In this, given σ, we say that event $e_i \in E(\sigma)$ is *earlier than* event $e_j \in E(\sigma)$ if $pos_\sigma(e_j) < pos_\sigma(e_j)$. The proof is easy as a direct application of Definitions 6 and 7.

Lemma 1 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions. We have that $\mathcal{I} \subseteq E(\sigma)$ is an ideal if and only if for all $e_i \in \mathcal{I}$ with $e_i = (a_i, k_i)$ we have that:*

- *if a_i is an input then $\mathcal{I}$ contains all earlier $e_j = (a_j, k_k)$ such that a_j is an input; and*
- *if a_i is an output then $\mathcal{I}$ contains all $e_j = (a_j, k_k)$ that are earlier than e_i.*

The following classical result [11] relates ideals and anti-chains.

Proposition 2 *The set of ideals is isomorphic to the set of anti-chains, by associating with every anti-chain E' the ideal which is the union of the principal ideals generated by the elements of E'. Vice versa, the anti-chain corresponding to a given ideal $\mathcal{I}$ is the set of maximal elements of $\mathcal{I}$.*

The following result [22] provides a measure, in the worst case, on the number of ideals contained in a set of events. This result will be relevant since it will be used to calculate the complexity of the algorithm that computes the automaton associated with a certain property P.

Proposition 3 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions with length m. There are $O(m^2)$ ideals in $E(\sigma)$.*

Proof By Proposition 2 we know that the number of ideals is the same as the number of anti-chains. However, we also know that any two inputs in $E(\sigma)$ are related under $\ll$. Similarly, any two outputs in $E(\sigma)$ are related under $\ll$. Thus, an anti-chain can have at most two elements (one input and one output) and so there are $O(m^2)$ anti-chains. The result therefore holds.

This result shows that there is a quadratic upper bound (in terms of the number of actions in a trace) and one might wonder whether the number of ideals really can be quadratic. Consider the following class of traces of length $2m$ in which input and output alternate: $\sigma_0 = \epsilon$; and for $m > 0$, $\sigma_m = \sigma_{m-1}?i_m!o_m$. Thus, for example, $\sigma_2 = ?i_1!o_1?i_2!o_2$. Each pair of the form $(?i_x, !o_y)$ with $y < x$ defines an anti-chain of $E(\sigma_m)$. Further, these are distinct anti-chains and so there must be at least one ideal for each pair $(?i_x, !o_y)$ with $y < x$. It is now sufficient to observe that there are $O(m^2)$ pairs (x, y) with $0 < y < x \leq m$.

4 Creating automata for properties

As explained in Section 2, we will consider properties of the form (σ, O_σ) for $\sigma \in \mathcal{A}ct^*$ and $O_\sigma \subseteq O$. Such a property says that if the SUT produces the sequence σ then the next output must come from the set O_σ. In our previous work [22] we defined an automaton $\mathcal{A}(\sigma, O_\sigma)$ accepting those traces that might have resulted from a trace of the SUT that does not satisfy the property (σ, O_σ). It is worth pointing out that the automaton does not produce false positives, but if an observed trace is accepted by the automaton then we cannot ensure that the trace produced by the SUT did not satisfy the property (σ, O_σ). In this section we adapt the above approach for the case where we know message latency bounds and take advantage of the timestamps included in the observations. We will show how, for trace σ, we can produce an automaton $\mathcal{A}^T_\Delta(\sigma)$ that accepts the set of timed traces that might be observed if the SUT produces σ. Since we are applying passive testing, a trace σ of interest might not be the start of the overall trace observed and so we will then adapt $\mathcal{A}^T_\Delta(\sigma)$ to produce the final automaton $\mathcal{A}_\Delta(\sigma, O_\sigma)$ that will be used for analysing traces. We use the ideals of $(E(\sigma), \ll)$ to represent states and based on this we define the transitions of a finite automaton $\mathcal{A}^T_\Delta(\sigma)$ that accepts the set of sequences in $\mathcal{L}_\Delta(\sigma)$. Transitions will include restrictions over the observed times of the actions in σ. These conditions will allow us to determine if the actions were performed at an instant *compatible* with the production of the considered trace by the SUT. It is worth noting that we do not require the expressive power of complex formalisms such as timed automata [2] to analyse the observed traces. In particular, we do not need to express relations between timed behavior in different states of the machine as can be done with timed automata by defining constraints involving different clocks that are initialized at different points. Our restrictions will be based on the latest and earliest times that an event might have been produced/observed. Therefore, our automaton will include a set of variables, associated with the actions belonging to σ, to store time information. These variables will be used to express the timed conditions that must be fulfilled by the actual timestamps of the traces.

Definition 8 Given a set X of variables, a *valuation* over X is a function that assigns a non-negative real number to every variable in X. The set of valuations of X, denoted $\mathcal{V}_X$, is the set of all the total functions from X to Time. Given a valuation $v \in \mathcal{V}_X$, a time $t \in$ Time and a set of variables $A \subseteq X$ we define the valuation $v[A/t]$ such that for all $x \in X$:

$$v[A/t](x) = \begin{cases} t & \text{if } x \in A \\ v(x) & \text{otherwise} \end{cases}$$

Given a set X of variables, the set $\mathcal{C}_X$ of constraints over X is defined by the following EBNF:

$$C ::= C \wedge C | E \circledcirc t$$
$$E ::= x + y | x - y$$

where $x, y \in X, t \in \text{Time}$ and $\circledcirc$ is an operator in $\{<, >, \leq, \geq\}$. The satisfaction by a valuation $v \in \mathcal{V}_X$ of the constraint C, denoted by $v \models C$, is defined as follows:

$$\begin{array}{ll} v \models \texttt{true} & \\ v \models C_1 \wedge C_2 & \text{iff } v \models C_1 \wedge v \models C_2 \\ v \models (x+y) \circledcirc t & \text{iff } (v(x)+v(y)) \circledcirc t \\ v \models (x-y) \circledcirc t & \text{iff } (v(x)-v(y)) \circledcirc t \end{array}$$

Finally, we use $\mathcal{P}(X)$ to denote the powerset of X.

Next we introduce the notion of an extended finite automaton. Essentially, this is a finite automaton extended with a set of variables; these variables will be used to impose conditions to trigger transitions.

Definition 9 Let S be a finite set of states, $\mathcal{A}ct$ be a set of actions, X be a finite set of variables, $s_I, s_F \in S$ be the initial and final states, and $Tr \subseteq S \times \mathcal{A}ct \times X \times \mathcal{C}_X \times \mathcal{P}(X) \times S$ be a set of transitions. We say that the tuple $A = (S, \mathcal{A}ct, Tr, X, s_I, s_F)$ is an *extended finite automaton*.

A *configuration* of A is a pair (s, v) where $s \in S$ is the current state and $v \in \mathcal{V}_X$ is the valuation corresponding to the current value of the variables belonging to X.

Given a configuration (s, v), if an action a is received at time t_a then a transition (s, a, t, C, Y, s') can be fired if $v[\{t\}/t_a] \models C$. In this case, the configuration will change to (s', v') where v' is the valuation such that

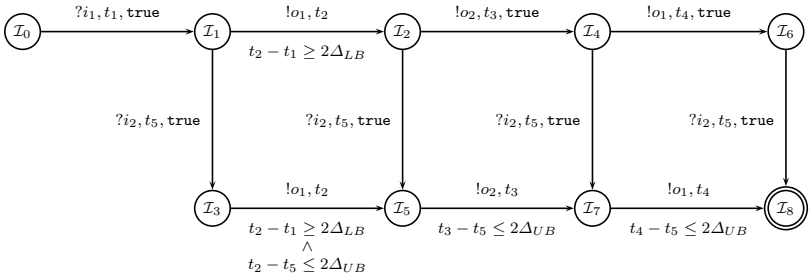
$$v'(x) = \begin{cases} t_a & \text{if } x = t \wedge v(t) = 0 \\ 0 & \text{if } x \in Y \\ v(x) & \text{otherwise} \end{cases}$$

The *initial valuation* of A, denoted by v_0, assigns 0 to every variable belonging to X.

Intuitively, a transition (s, a, t, C, Y, s') will be fired if the current state is s, the automaton receives the action a and the condition C holds for the current valuation of the variables. If the transition is triggered then the variables belonging to Y are reset to zero, the variable t is updated to the time when the action was received and the current state becomes s'. Next we define the first type of automata that we use in our approach. Essentially, given $\Delta = (\Delta_{LB}, \Delta_{UB})$ and a sequence of actions σ, the extended finite automaton $\mathcal{A}^T_\Delta(\sigma)$ will accept those traces that are a feasible variation of σ taking into account that the time difference between the instant when the input was observed at the monitor and was received by the SUT (and symmetrically for outputs) is bounded between Δ_{LB} and Δ_{UB}.

Definition 10 Let $\sigma = a_1 \ldots a_n \in \mathcal{A}ct^*$ be a non-empty sequence of actions and $\Delta = (\Delta_{LB}, \Delta_{UB}) \in \text{Time} \times \text{Time}$. The *extended finite automaton for σ and Δ*, denoted by $\mathcal{A}^T_\Delta(\sigma)$, is defined as $(S, \mathcal{A}ct, Tr, X, \mathcal{I}_s, \mathcal{I}_f)$ where

- S, the set of states, is equal to the set of ideals of $(E(\sigma), \ll)$.
- $\mathcal{A}ct$ is the alphabet.
- $X = \{t_j | 1 \leq j \leq n\}$ is a finite set of variables.
- $\mathcal{I}_s = \{\}$ is the initial, or start, state.
- $\mathcal{I}_f = E(\sigma)$ is the final state.



$\mathcal{I}_0 = \{\}$
$\mathcal{I}_1 = \{(?i_1, 1)\}$
$\mathcal{I}_2 = \{(?i_1, 1), (!o_1, 1)\}$
$\mathcal{I}_3 = \{(?i_1, 1), (?i_2, 1)\}$
$\mathcal{I}_4 = \{(?i_1, 1), (!o_1, 1), (!o_2, 1)\}$
$\mathcal{I}_5 = \{(?i_1, 1), (!o_1, 1), (?i_2, 1)\}$
$\mathcal{I}_6 = \{(?i_1, 1), (!o_1, 1), (!o_2, 1), (!o_1, 2)\}$
$\mathcal{I}_7 = \{(?i_1, 1), (!o_1, 1), (!o_2, 1), (?i_2, 1)\}$
$\mathcal{I}_8 = \{(?i_1, 1), (!o_1, 1), (!o_2, 1), (!o_1, 2), (?i_2, 1)\}$

Fig. 4 Automaton $\mathcal{A}^T_\Delta(\sigma)$ for the trace $\sigma = ?i_1!o_1!o_2!o_1?i_2$.

and a tuple $(\mathcal{I}, a, t_i, C, Y, \mathcal{I}')$ belongs to the set of transitions Tr if and only if there exists an event $(a, k) \in (E(\sigma), \ll)$ such that $\mathcal{I}' = \mathcal{I} \cup \{(a, k)\}$ and $pos_\sigma((a, k)) = i$, $Y = \emptyset$ and the restriction C is defined by the sequential application of the following three rules:

1. $C := \texttt{true}$
2. If $1 < i \leq n \wedge a_i \in O \wedge a_{i-1} \in I$ then
$$C := C \wedge t_i - t_{i-1} \geq 2\Delta_{LB}$$
3. If $1 \leq i < n \wedge a_i \in O \wedge \exists i < j \leq n$ such that $a_j \in I \wedge (a_j, k_j) \in \mathcal{I}$ then
$$C := C \wedge t_i - t_{min\{j | a_j \in I \wedge (a_j, k_j) \in \mathcal{I} \wedge j > i\}} \leq 2\Delta_{UB}$$

Given a property (σ, O_σ) the automaton $\mathcal{A}^T_\Delta(\sigma)$ accepts the timestamped traces that might be observed if the SUT produces the trace σ in an asynchronous setting where message latency is given by Δ. The transitions of the automaton include temporal conditions in order to ensure that the relation between the different timestamps is *compatible* with the production of σ. In this task, it is necessary to take into account the considered message latency. We assume that the supplied timestamped trace corresponds to a timed trace, that is, we cannot have two consecutive actions where the timestamp of the first one is greater than or equal to the timestamp of the latter one. However, it is not difficult to extend the definition to include a condition attached to all the transitions to ensure that the traces do actually satisfy this restriction. Each of the n variables t_i with $1 \leq i \leq n$, will store the observed time corresponding to the i-th action of the trace σ. Next we briefly explain how temporal restrictions appearing in transitions are defined. Intuitively, their purpose is to capture the causality relations among the actions included in σ. Only transitions labelled by an output may have a non-trivial, that is, different from `true`, associated condition. The reason is that we only need to check whether the delay of output actions due to the latency of the asynchronous

setting might correspond to the excepted trace. Step 2 in the construction of the temporal restriction is used to discard those traces showing an output $!o$ after an input $?i$, as initially expected, but such that the timestamps together with the lower bound in Δ, indicate that $!o$ was produced before $?i$. Step 3 also deals with conditions on the observation of an output after an input, but assuming that the property indicates that the input must be received after the output is produced. In this case, the restriction associated with the transition allows us to determine whether the timestamps are consistent with the production of the output after the reception of the input, in order to discard the observed prefix of the trace. Finally, let us note that although the set of variables to be reset appearing in all the transitions of the automaton $\mathcal{A}^T_\Delta(\sigma)$ is always equal to the empty set, that is, no variable is set to 0, this element has been included in the definition of transitions because it will be used, with different values, in the adapted version of the automaton that is presented later.

Next, we formally define the meaning of an automaton $\mathcal{A}^T_\Delta(\sigma)$ accepting a timed trace.

Definition 11 Let $\rho = (d_1, r_1) \dots (d_n, r_n) \in (\mathcal{A}ct \times \text{Time})^*$, $\sigma \in \mathcal{A}ct^*$ and $\Delta \in \text{Time} \times \text{Time}$. We say that the automaton $\mathcal{A}^T_\Delta(\sigma)$ *accepts* ρ if there exist n transitions $(\mathcal{I}_0, d_1, t_1, C_1, \emptyset, \mathcal{I}_1), \dots, (\mathcal{I}_{n-1}, d_n, t_n, C_n, \emptyset, \mathcal{I}_n)$, where $\mathcal{I}_0 = \mathcal{I}_s$ and $\mathcal{I}_n = \mathcal{I}_f$, that can be fired for the configurations $(\mathcal{I}_0, v_0), \dots, (\mathcal{I}_{n-1}, v_{n-1})$, respectively, where for all $1 \leq j \leq n$ we have that v_j is the valuation obtained after the transition $(\mathcal{I}_{j-1}, d_j, t_j, C_j, \emptyset, \mathcal{I}_j)$ is executed. The set of all the timed traces accepted by $\mathcal{A}^T_\Delta(\sigma)$ is denoted by $L(\mathcal{A}^T_\Delta(\sigma))$.

As previously stated, the automaton $\mathcal{A}^T_\Delta(\sigma)$ accepts those sequences of pairs (action,time) such that the actions label walks from its initial state to its final state and the time values satisfy the constraints imposed by the transitions. Next, we give an example.

Example 7 Let $\sigma = ?i_1!o_1!o_2!o_1?i_2$ be a trace. Figure 4 depicts the automaton $\mathcal{A}^T(\sigma)$ that accepts the set of sequences in $\mathcal{L}_\Delta(\sigma)$, where we have omitted the sets of variables that will be reset in the each transition because they are always empty.

Next, we show how some of the transitions have been obtained. The transition $(\mathcal{I}_0, ?i_1, t_1, \texttt{true}, \emptyset, \mathcal{I}_1)$ is included in the automaton because $\mathcal{I}_1 = \mathcal{I}_0 \cup \{(?i_1, 1)\}$ and the condition is equal to `true` because the action is an input. The transition $(\mathcal{I}_1, !o_1, t_2, t_2 - t_1 \geq 2\Delta_{LB}, \mathcal{I}_2)$ corresponds to the observation of the second action in σ, that is $!o_1$, after the first one, that is $?i_1$. In this case, the restriction will be used to determine whether it is possible, based on the timestamps, that $!o_1$ was produced after $?i_1$ was received. If the condition is not satisfied then we know that $!o_1$ was sent before the reception of $?i_1$ and we will not proceed. The transition $(\mathcal{I}_3, !o_1, t_2, t_2 - t_1 \geq 2\Delta_{LB} \wedge t_2 - t_5 \leq 2\Delta_{UB}, \mathcal{I}_5)$ is associated with the possibility that the observation of output $!o_1$ is delayed until after the input $?i_2$ in σ has been sent. In this situation, it is necessary to check that the output was not produced neither before $?i_1$ was received $(t_2 - t_1 \geq 2\Delta_{LB})$ nor after $?i_2$ was received $(t_2 - t_5 \leq 2\Delta_{UB})$.

The next result shows that our automaton captures the admissible variations in the observation of a trace σ for a given Δ.

Proposition 4 *Given* $\sigma = a_1 \dots a_n \in \mathcal{A}ct^*$ *and* $\Delta = (\Delta_{LB}, \Delta_{UB}) \in \text{Time} \times \text{Time}$, *we have that* $L(\mathcal{A}^T_\Delta(\sigma)) = \mathcal{L}_\Delta(\sigma)$.

Proof We will prove a slightly stronger result, which is that ρ is a sequence of pairs (action,time) that labels a walk from the initial state of $\mathcal{A}^T_\Delta(\sigma)$ where the time values satisfy the timed constraints imposed by the transitions if and only if ρ is a prefix of a sequence in $\mathcal{L}_\Delta(\sigma)$.

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, let us assume that the result holds for all sequences of length less than k, $k \geq 1$, and ρ has length k. Thus, $\rho = \rho_1(a_j, t)$ for some $1 \leq j \leq n$ and $t \in \text{Time}$. By the inductive hypothesis we have that ρ_1 is a prefix of a sequence in $\mathcal{L}_\Delta(\sigma)$. In addition, by the definition of $\mathcal{A}^T_\Delta(\sigma)$, we have that the set of actions in ρ_1 forms an ideal $\mathcal{I}_1$ and there exists $1 \leq k \leq n$ such that $\mathcal{I}_1 \cup \{(a_j, k)\}$ is an ideal. Thus, since $\mathcal{I}_1 \cup \{(a_j, k)\}$ is an ideal, there does not exist $(b, k') \in E(\sigma) \setminus (\mathcal{I}_1 \cup \{(a_j, k)\})$ such that $b \ll a_j$. By the definition of $\mathcal{L}_\Delta(\sigma)$ we have that ρ_1 can be followed by (a_j, t) for some $t \in \text{Time}$ if some timed conditions are fulfilled. We must consider different cases:

- $a_j \in O$ and $a_{j-1} \in I$. In this case $a_{j-1} \ll a_j$. We have that $\mathcal{I}_1 \cup \{(a_j, k)\}$ forms an ideal. Therefore, $a_{j-1} \in \mathcal{I}_1$ and $\rho_1 = \rho'(a_{j-1}, t')\rho''$ for some $t' \in \text{Time}$ and sequences ρ' and ρ''. By the construction of $\mathcal{A}^T_\Delta(\sigma)$, we have that $t - t' \geq 2\Delta_{LB}$ as required by the definition of $\mathcal{L}_\Delta(\sigma)$.
- $a_j \in O$ and $a_{j+1} \in I$. In this case the definition of $\mathcal{L}_\Delta(\sigma)$ establishes conditions over the observation time values associated with both actions if a_j is observed after a_{j+1}. If this is the case, we must have that $a_{j+1} \in \mathcal{I}_1$. Therefore, $\rho_1 = \rho'(a_{j+1}, t')\rho''$ for some $t' \in \text{Time}$ and sequences ρ' and ρ''. By the definition of $\mathcal{A}^T_\Delta(\sigma)$, we have that $t - t' \leq 2\Delta_{UB}$ as required by the definition of $\mathcal{L}_\Delta(\sigma)$.

Finally, we have to show that each timed observation in $\mathcal{L}_\Delta(\sigma)$ must be a timed trace. Therefore, the timestamps associated with the actions must follow an increasing order. The construction of $\mathcal{A}^T_\Delta(\sigma)$ establishes for all the transitions that the observation time value of an action must be greater than the previous one. Therefore, ρ is a timed trace. Then, $\rho_1(a_j, t)$ is a prefix of a sequence in $\mathcal{L}_\Delta(\sigma)$ as required.

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, let us assume that it holds for all sequences of length less than k, $k \geq 1$, and ρ has length k. Thus, $\rho = \rho_1(a_j, t)$ for some $1 \leq j \leq n$ and $t \in \text{Time}$. By the inductive hypothesis we have that ρ_1 is the label of a walk of $\mathcal{A}^T_\Delta(\sigma)$ where the time values satisfy the timed constraints imposed by the transitions and we assume that this walk reaches a state representing ideal $\mathcal{I}_1$. By the definition of $\mathcal{L}_\Delta(\sigma)$ there cannot exist an action in σ that is not in ρ_1 and that must be observed before a_j and so precedes a_j under $\ll$. Thus, $\mathcal{I}_2 = \mathcal{I}_1 \cup \{(a_j, k_j)\}$ for some $1 \leq k_j \leq n$ is an ideal and so $\mathcal{A}^T_\Delta(\sigma)$ contains a transition from the state representing $\mathcal{I}_1$ to the state representing $\mathcal{I}_2$ with label a_j. We need to prove that the timed restrictions associated with this transition are satisfied by the valuation at state $\mathcal{I}_1$ when the time values of the trace ρ are considered.

By the definition of $\mathcal{L}_\Delta(\sigma)$, if $a_j \in O$, $a_{j-1} \in I$ and $\rho_1 = \rho'(a_{j-1}, t')\rho''$ for some $t' \in \text{Time}$ and sequences ρ' and ρ'', we have that $t - t' \geq 2\Delta_{LB}$. By the construction

of $\mathcal{A}^T_\Delta(\sigma)$, this restriction is associated with the transition departing from the state representing $\mathcal{I}_1$ and reaching the state representing $\mathcal{I}_2$ with label a_j. In the same way, if $a_j \in O$, $a_{j+1} \in I$, and a_{j+1} is observed before a_j, then the definition of $\mathcal{L}_\Delta(\sigma)$ establishes that $t - t' \leq 2\Delta_{UB}$. Again, the construction of $\mathcal{A}^T_\Delta(\sigma)$ associates this condition with each transition originating from a state representing an ideal $\mathcal{I}$ and reaching the state representing an ideal $\mathcal{I}'$ with label a_j if $a_{j+1} \in \mathcal{I}$.

We conclude that $\rho = \rho_1(a_j, t)$ is the label of a walk of $\mathcal{A}^T_\Delta(\sigma)$ where the time values satisfy the timed constraints imposed by the transitions as required. The result therefore follows.

We have that $\mathcal{A}^T_\Delta(\sigma)$ defines the set of behaviours included in $\mathcal{L}_\Delta(\sigma)$ that can be observed if the SUT produces σ. Now, we have to adapt $\mathcal{A}^T_\Delta(\sigma)$ to take into account the fact that a trace of interest might not be the start of the overall trace observed, that is, σ might be preceded by other actions and the observation of earlier outputs might be delayed. In addition, σ might be followed by later actions and the outputs from σ might not be observed until after later inputs. We will then modify $\mathcal{A}^T_\Delta(\sigma)$ to produce the extended finite automaton $\mathcal{A}_\Delta(\sigma, O_\sigma)$ that will be used. Algorithm 1 achieves this. Taking $\mathcal{A}^T_\Delta(\sigma)$ as the starting point, we add new transitions to deal with actions that could be interleaved with the observation of σ. After the initialisation, the first step (4) adds self-loops to the initial state to capture all the possible starting points in the observed trace. Next, in order to consider the possible delay of earlier outputs, new self-loops labelled by all possible output actions are added to the automaton in states corresponding to ideals that only contain input actions (5). The fact that we consider a FIFO communications mechanism requires that earlier outputs can be delayed only until before the first output of σ is observed. Actually, the temporal conditions attached to the new transitions allow us to check that only outputs that might be produced before the first input action of σ are accepted. Once all the inputs of σ have been observed it is possible that the unobserved outputs in σ are delayed after later inputs. Therefore, we need to include new self-loops labelled by all possible inputs in states containing all the inputs in σ (6). The instant when the first later input is observed by the monitor will be stored in the variable t_{ei}. It will be used to check that the outputs in σ observed after that moment might correspond to a possible delay. In order to do this, the temporal restrictions associated with transitions labelled by an output action and outgoing from states corresponding to ideals that contain all the inputs in σ are extended with new temporal constraints. If the final state of the original automaton is reached then all the actions from σ have been observed. If the next observed output does not belong to O_σ then the SUT does not satisfy the property. In order to capture this fact, a new transition leading to an error state s_e is included in the automaton (7). This will be the only final state of our automaton. Finally, each of the states will be completed with transitions that will capture all the possible combinations of actions and conditions not considered in the transitions outgoing from it (8). These transitions correspond to observations that do not fit with the production of σ by the SUT, except the one outgoing from the state s_f that capture the observation of an output action in O_σ after the observation of a trace in $\mathcal{L}_\Delta(\sigma)$. All these transitions will reset all the variables and lead to the initial state.

Next we give an example to illustrate the transformation from the initial automaton $\mathcal{A}^T_\Delta(\sigma)$ to the final one $\mathcal{A}_\Delta(\sigma, O_\sigma)$.

Algorithm 1 Producing $\mathcal{A}_{\Delta}(\sigma, O_\sigma)$

1: Input (σ, O_σ).
2: Let $\mathcal{A}^T_{\Delta}(\sigma) = (S, \mathcal{A}ct, Tr, X, \mathcal{I}_s, \mathcal{I}_f)$.
3: Let $\mathcal{A}_{\Delta}(\sigma, O_\sigma) = (S \cup \{s_e\}, \mathcal{A}ct, Tr, Z, s_0, s_e)$ where $Z = X \cup \{t_x, t_{ei}\}$, $s_0 = \mathcal{I}_s$, and $s_e \notin S$ is a fresh state.
4: For all $a \in \mathcal{A}ct$ add the transition $(s_0, a, t_x, \texttt{true}, \varnothing, s_0)$. *These transitions ensure that we are considering all possible starting points in a trace ρ' observed.*
5: For every state s of $\mathcal{A}_{\Delta}(\sigma, O_\sigma)$ that represents an ideal that does not contain output and for all $!o \in O$, add the transition $(s, !o, t_x, t_x - t_k \leq 2\Delta_{UB}, \varnothing, s)$ where $t_k \in X$ is the variable that corresponds to the first input in σ. *These transitions correspond to the possibility of earlier output being observed after input from σ. The constraints associated with these transitions require these outputs not to be produced after the first input in σ.*
6: For every state s of $\mathcal{A}_{\Delta}(\sigma, O_\sigma)$ that represents an ideal that contains all of the inputs from σ and for all $?i \in I$, add the transition $(s, ?i, t_{ei}, \texttt{true}, \varnothing, s)$. In addition, for each transition $(s, !o, t, C, Y, s')$ outgoing from a state of $\mathcal{A}_{\Delta}(\sigma, O_\sigma)$ that represents an ideal that contains all of the input from σ, we convert the condition C to $C \wedge ((t_{ei} > 0 \wedge t - t_{ei} \leq 2\Delta_{UB}) \vee ((t_{ei} = 0))$. *These transitions correspond to the possibility of later input being observed before some of the output from σ. The time corresponding to the first later input observed is stored in the variable t_{ei} and the constraints associated with the transitions outgoing from states that contains all the inputs and are labelled by outputs are extended with a new requirement to ensure that the output correspond to a possible delay. Note that by our definition of an extended finite automaton, the value of t_{ei} is only changed if it is zero and so t_{ei} will record the time of the first 'additional' input even if more are received.*
7: For all $!o \in O \setminus O_\sigma$ add the transition $(\mathcal{I}_f, !o, t_x, \texttt{true}, \varnothing, s_e)$. *If we have observed a feasible variation of the inputs and outputs from σ and the next output is not from O_σ then go to the final (error) state.*
8: Complete $\mathcal{A}_{\Delta}(\sigma, O_\sigma)$: For all $a \in \mathcal{A}ct$ and state s, let us consider the set of transitions $Tr_a = \{(s, a, t, C, Y, s') | s' \in S \wedge t \in Z\}$. If $\bigvee_{tr \in Tr_a} C \not\equiv \texttt{true}$ then add the transition (s, a, t_x, C_a, Z, s_0), where $C_a \equiv \neg \bigvee_{tr \in Tr_a} C$
9: Output $\mathcal{A}_{\Delta}(\sigma, O_\sigma)$.

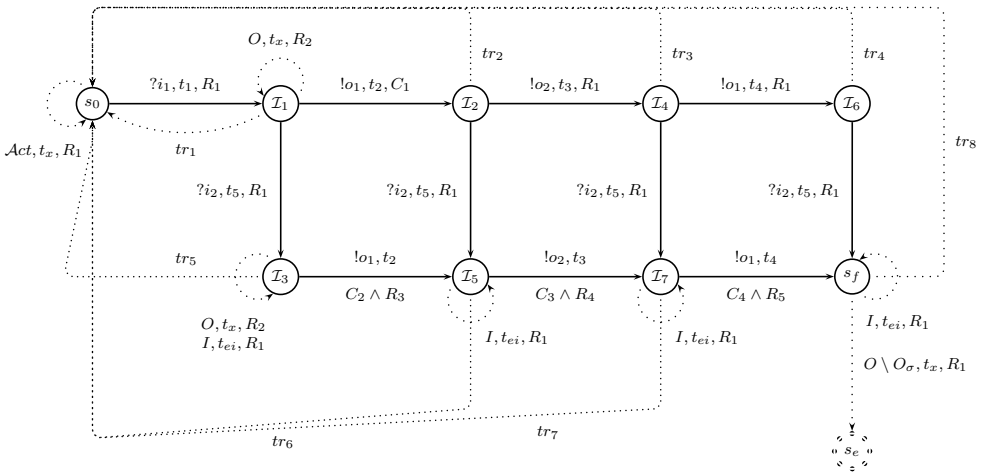
Example 8 Let $\sigma = ?i_1 !o_1 ?i_2 !o_1 !o_2 ?i_2$ and let us consider the automaton $\mathcal{A}^T_{\Delta}(\sigma)$ depicted in Figure 4. Given a set of outputs O_σ and Δ, Figure 5 shows the extended finite automaton $\mathcal{A}_{\Delta}(\sigma, O_\sigma)$ constructed by using Algorithm 1.

We will show that Algorithm 1 produces an automata that adequately encodes the characteristics of its associated property. First, we recall our notion of *sound* automaton [22].

Definition 12 Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions, $\Delta \in \text{Time} \times \text{Time}$ be a pair of bounds and $O_\sigma \subseteq O$ be a set of outputs. Let us consider the property $P = (\sigma, O_\sigma)$. We say that an extended finite automaton A is *sound for* P and Δ if and only if whenever the SUT produces a trace σ_1 that does not satisfy property P and the timed trace $\sigma'_1 \in \mathcal{L}_{\Delta}(\sigma_1)$ is observed we have that $\sigma'_1 \in L(A)$.

This notion essentially corresponds to the automaton not being able to produce false positives. The following result proves that the automaton produced by Algorithm 1 is sound.

Theorem 1 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions, $\Delta \in \text{Time} \times \text{Time}$ be a pair of bounds and $O_\sigma \subseteq O$ be a set of outputs. Given the property $P = (\sigma, O_\sigma)$, the automaton $\mathcal{A}_{\Delta}(\sigma, O_\sigma)$ returned by Algorithm 1 is sound for P and Δ.*



$R_1 = \texttt{true}$
$R_2 = t_x - t_1 \geq 2\Delta_{UB}$
$R_3 = (t_{ei} > 0 \wedge t_2 - t_{ei} \leq 2\Delta_{UB}) \vee t_{ei} = 0)$
$R_4 = (t_{ei} > 0 \wedge t_3 - t_{ei} \leq 2\Delta_{UB}) \vee t_{ei} = 0)$
$R_5 = (t_{ei} > 0 \wedge t_4 - t_{ei} \leq 2\Delta_{UB}) \vee t_{ei} = 0)$
$tr_1 = \{(I \setminus \{?i_2\}, t_x, \texttt{true}), (O \setminus \{!o_1\}, t_x, \neg R_2), (!o_1, t_x, \neg C_1 \wedge \neg R_2)\}$
$tr_2 = \{(O \cup I \setminus \{?i_2, !o_2\}), t_x, \texttt{true})\}$
$tr_3 = \{(O \cup I \setminus \{?i_2, !o_1\}), t_x, \texttt{true})\}$
$tr_4 = \{(I \setminus \{?i_2\}, t_x, \texttt{true})\}$
$tr_5 = \{(O \setminus \{!o_1\}, t_x, \neg R_2), (!o_1, t_x, (\neg C_2 \vee \neg R_3) \wedge \neg R_2\}$
$tr_6 = \{(O \setminus \{!o_2\}, t_x, \texttt{true}), (!o_2, t_x, \neg C_3 \vee \neg R_4\}$
$tr_7 = \{(O \setminus \{!o_1\}, t_x, \texttt{true}), (!o_1, t_x, \neg C_4 \vee \neg R_5\}\}$
$tr_8 = \{(O_\sigma, t_x, \texttt{true})\}$

Fig. 5 Automaton $\mathcal{A}_\Delta(\sigma, O_\sigma)$ for the trace $\sigma = ?i_1!o_1!o_2!o_1?i_2$ and the set of outputs O_σ.

Proof Let us recall that we label actions using their occurrence, if necessary, so that they are unique and this labelling is preserved by the delay of output. We assume that the SUT has produced a trace σ_1 that does not satisfy P, and it led to the observation of the timed trace $\sigma_1' \in \mathcal{L}_\Delta(\sigma_1)$ and we are required to prove that $\sigma_1' \in L(\mathcal{A}_\Delta(\sigma, O_\sigma))$. Since σ_1 does not satisfy P we have that $\sigma_1 = \sigma_2 \sigma a \sigma_3$ for some $a \in O \setminus O_\sigma$. Since $\sigma_1' \in \mathcal{L}_\Delta(\sigma_1)$ we have that $\sigma_1' = \sigma_2' \sigma'(a, t_a)\sigma_3'$ for some $\sigma', \sigma_2', \sigma_3' \in (Act \times Time)^*$ and $t_a \in Time$ such that σ' satisfies the following.

- σ' starts with the first action of σ_1' that is from σ;
- σ' may contain outputs not in σ (delayed from σ_2);
- σ' may contain inputs not in σ (due to outputs from σ or a being delayed past inputs from σ_3); and
- σ' can have outputs from σ being delayed past inputs from σ.

By the definition of $\mathcal{A}_\Delta(\sigma, O_\sigma)$ we have that the state of $\mathcal{A}_\Delta(\sigma, O_\sigma)$ after σ_2' can be the initial state of $\mathcal{A}_\Delta(\sigma, O_\sigma)$. Further, by Proposition 4 we know that σ' with the extra initial outputs and final inputs removed can take $\mathcal{A}_\Delta(\sigma, O_\sigma)$ to the final state s_f. Consider the corresponding path ρ of $\mathcal{A}_\Delta(\sigma, O_\sigma)$.

The additional outputs in σ' that are not in σ (and so come from σ_2) are all before the first output in σ' that was from σ and so, by construction, we can define a path ρ' that includes these by adding self-loops to ρ with timed restrictions requiring the outputs were not produced after the first input from σ.

Similarly, the additional inputs in σ' that are not in σ (and so come from σ_3) are all after the last input in σ' that was from σ and so we can define a path ρ'' that includes these by adding self-loops to ρ' with timed restrictions requiring these inputs were not produced before the first output from σ. Path ρ'' thus takes $\mathcal{A}(\sigma, O_\sigma)$ to state s_f and has label σ'. The result now follows from observing that a takes $\mathcal{A}(\sigma, O_\sigma)$ from state s_f to the final state and this final state cannot be left.

An automaton A being sound for P denotes an absence of false positives. However, we might also want the absence of false negatives: if the SUT produces a trace and the resultant observation is in $L(A)$ then the trace produced by the SUT must not have satisfied P. This is captured by the notion of an *exact* automaton.

Definition 13 Let P be a property and $\Delta = (\Delta_{LB}, \Delta_{UB}) \in \text{Time} \times \text{Time}$ be a pair of bounds. We say that an extended finite automaton A is *exact for* P and Δ if and only if whenever the SUT produces some trace σ_1 and the observed trace $\sigma_1' \in \mathcal{L}_\Delta(\sigma_1)$ is such that $\sigma_1' \in L(A)$ we must have that σ_1 does not satisfy P.

Unfortunately, the automaton $\mathcal{A}_\Delta(\sigma, O_\sigma)$, produced by our algorithm, need not be exact for (σ, O_σ) as the following example shows.

Example 9 Let us consider the property $P = (?i, \{!o\})$, the bounds $\Delta = (0.1, 0.3)$ and the observed trace $\sigma_1' = (?i, 0.2)(!o', 0.7)(!o, 1.6)$. This trace belongs to the set $L(\mathcal{A}_\Delta(?i, \{!o\}))$. In this situation we have two admissible possibilities for the trace σ_1 actually produced by the SUT.

- $\sigma_1 = !o'?i!o$ and so σ_1 satisfies P.
- $\sigma_1 = ?i!o'!o$ and so σ_1 does not satisfy P.

Therefore, we cannot categorically conclude that the SUT does not fulfill the property.

The above shows that an observation made might be consistent with both traces that satisfy P and traces that do not. Therefore, our automata might not be exact. Despite this, we would like to ensure that if the observed trace is in $L(A)$ then the actual trace produced by the SUT might be one that does not satisfy property P. This is captured by the following notion.

Definition 14 Let P be a property and $\Delta = (\Delta_{LB}, \Delta_{UB}) \in \text{Time} \times \text{Time}$ be a pair of bounds. We say that an extended finite automaton A is *precise for* P if and only if whenever a trace σ_1' is in $L(A)$ there is some trace σ_1 that does not satisfy P such that $\sigma_1' \in \mathcal{L}_\Delta(\sigma_1)$.

The following result shows that Algorithm 1 returns an automaton that is precise for the considered property.

Theorem 2 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions, $\Delta \in \text{Time} \times \text{Time}$ be a pair of bounds and $O_\sigma \subseteq O$ be a set of outputs. Given the property $P = (\sigma, O_\sigma)$, the automaton $\mathcal{A}_\Delta(\sigma, O_\sigma)$ returned by Algorithm 1 is precise for P and Δ.*

Proof Let us suppose that trace σ'_1 is in $L(\mathcal{A}_\Delta(\sigma, O_\sigma))$. By definition it is sufficient to prove that there exists some trace σ_1 that does not satisfy P such that $\sigma'_1 \in \mathcal{L}_\Delta(\sigma_1)$. Note that σ_1 does not satisfy P if and only if it has a prefix that ends in σa for some $a \in O \setminus O_\sigma$.

By the construction of $\mathcal{A}_\Delta(\sigma, O_\sigma)$, since $\sigma'_1 \in L(\mathcal{A}_\Delta(\sigma, O_\sigma))$, we have that σ'_1 has prefix $\sigma'_2\sigma'_3(a,t)$ for some $t \in$ Time such that σ'_3 takes $\mathcal{A}_\Delta(\sigma, O_\sigma)$ from state s_0 to s_f and $a \in O \setminus O_\sigma$. In addition, we must have that untime(σ'_3) differs from σ in only three ways:

- the addition of outputs before the outputs of σ, through self-loops in states that correspond to ideals that contain no output;
- the addition of inputs after the inputs of σ, through self-loops in states that correspond to ideals that contain all of the inputs from σ; and
- the delay in output from σ.

Thus, the input projection of untime(σ'_3) is the input projection of σ followed by some sequence σ'_I of inputs and the output projection of untime(σ'_3) is some sequence σ'_O of outputs followed by the output projection of σ. In addition, by the construction of the automaton, timed conditions must be fulfilled by the timestamps in σ'_3. On the one hand earlier outputs might have been produced before the first input in σ and outputs in σ being delayed might have been produced before the first later input. As a result, $\sigma'_3(a,t) \in \mathcal{L}_\Delta(\sigma'_O\sigma\sigma'_I a)$. Thus, σ'_1 has prefix $\sigma'_2\sigma'_3(a,t)$ such that $\sigma'_2\sigma'_3(a,t) \in \mathcal{L}_\Delta(\text{untime}(\sigma'_2)\sigma'_O\sigma\sigma'_I a)$ for some $a \in O \setminus O_\sigma$. By definition, since σ'_I contains only input actions and a is an output, we have that $\mathcal{L}_\Delta(\text{untime}(\sigma'_2)\sigma'_O\sigma\sigma'_I a) \subseteq \mathcal{L}_\Delta(\text{untime}(\sigma'_2)\sigma'_O\sigma a\sigma'_I)$. Therefore σ'_1 has prefix $\sigma'_2\sigma'_3(a,t)$ such that $\sigma'_2\sigma'_3(a,t) \in \mathcal{L}_\Delta(\text{untime}(\sigma'_2)\sigma'_O\sigma a\sigma'_I)$ for some $a \in O \setminus O_\sigma$. Since untime$(\sigma'_2)\sigma'_O\sigma a\sigma'_I$ does not satisfy property P, we can set $\sigma_1 = \text{untime}(\sigma'_2)\sigma'_O\sigma a\sigma'_I$ and the result follows.

By Proposition 3 we know that $\mathcal{A}(\sigma, O_\sigma)$ has $O(|\sigma|^2)$ states. In addition, we can construct the relation $\ll$ and the set of anti-chains in $O(|\sigma|^2)$ time. The following result is therefore immediate.

Proposition 5 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions, $\Delta \in$ Time $\times$ Time be a pair of bounds and $O_\sigma \subseteq O$ be a set of outputs. Given the property (σ, O_σ), the process of generating the automata $\mathcal{A}(\sigma, O_\sigma)$ takes $O(|\sigma|^2)$ time.*

In passive testing we can update the current state of $\mathcal{A}(\sigma, O_\sigma)$ whenever we make a new observation and thus the complexity of applying passive testing is linear in the length of the trace that the SUT is producing. The following shows that the process is polynomial in the length of σ, which suggests that it can be applied in real-time since $|\sigma|$ is usually relatively small.

Proposition 6 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions, $\Delta \in$ Time $\times$ Time be a pair of bounds and $O_\sigma \subseteq O$ be a set of outputs. Given the property (σ, O_σ), the process of updating the state of $\mathcal{A}(\sigma, O_\sigma)$ takes $O(|\sigma|^4)$ time when a new input or output is observed.*

Proof At each point in the process of simulating $\mathcal{A}(\sigma, O_\sigma)$ with a trace we have a current set of states. Let us suppose that a new action a is observed. It is well-known that the membership problem can be tested in time $O(n \cdot s^2)$, where n is

the length of the sequence and s is the number of states of the automaton [24]. Since we do this for one action ($n = 1$) and we know that $\mathcal{A}(\sigma, O_\sigma)$ has $O(|\sigma|^2)$ states, we conclude that the overall time complexity is in $O(|\sigma|^4)$ time.

5 Conclusions and future work

In this paper we presented a formal framework for the passive testing of software systems with asynchronous communications. This corresponds to the situation in which there is an asynchronous network between the monitor, which observes inputs and outputs, and the system under test (SUT). Due to the nature of the communications, there may be alternative traces of the SUT that are consistent with a trace σ observed by the monitor since the observation of an output $!o$ produced by the SUT might be delayed past the observation of an input $?i$ sent to the SUT. We considered the case where we can label an action with the time at which it was observed by the monitor and we have lower and upper bounds on the time that it takes for messages to pass between the monitor and the SUT. This allows us to deduce more about the actual order of events produced by the SUT. We developed an automata-based approach that checks whether the observations regarding the SUT contradict the requirement imposed by the monitor. Importantly, this approach operates in low-order polynomial time and requires little storage. As a result, the approach is suitable for use in real-time.

We have identified several lines for future work. First, we have developed a tool implementing the different formalisms and algorithms of the untimed framework [6]. We plan to extend it so that the new tool will help us to evaluate the framework presented in this paper. As a more theoretical line of work, we plan to extend our monitors to include other, more complex, safety properties. Finally, we would like to add probabilistic information on the time values associated with the observation of actions so that the induced probability distribution functions can provide additional information about the real causality, at the SUT, between the actions observed at the monitor.

References

1. Abreu, F.B., Morais, A.N.P., Cavalli, A.R., Wehbi, B., Montes de Oca, E., Mallouli, W.: An effective attack detection approach in wireless mesh networks. International Journal of Space-Based and Situated Computing **5**(2), 89–99 (2015)
2. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126**, 183–235 (1994)
3. Andrés, C., Merayo, M.G., Núñez, M.: Formal passive testing of timed systems: Theory and tools. Software Testing, Verification and Reliability **22**(6), 365–405 (2012)
4. Bayse, E., Cavalli, A., Núñez, M., Zaïdi, F.: A passive testing approach based on invariants: Application to the WAP. Computer Networks **48**(2), 247–266 (2005)
5. Berry, G., Sethi, R.: From regular expressions to deterministic automata. Theoretical Computer Science **48**(3), 117–126 (1986)
6. Camacho, M.A., Merayo, M.G., Medina-Bulo, I.: PTTAC: Passive Testing Tool for Asynchronous Systems. In: 10th Int. Conf. on Signal-Image Technology & Internet-Based Systems, SITIS'14, pp. 223–229. IEEE Computer Society (2014)
7. Cavalli, A., Gervy, C., Prokopenko, S.: New approaches for passive testing using an extended finite state machine specification. Information and Software Technology **45**(12), 837–852 (2003)

8. Cavalli, A.R., Higashino, T., Núñez, M.: A survey on formal active and passive testing with applications to the cloud. Annales of Telecommunications **70**(3-4), 85–93 (2015)
9. Che, X., Maag, S.: Passive performance testing of network protocols. Computer Communications **51**, 36–47 (2014)
10. Colombo, C., Pace, G.J., Abela, P.: Safer asynchronous runtime monitoring using compensations. Formal Methods in System Design **41**(3), 269–294 (2012)
11. Dilworth, R.P.: A decomposition theorem for partially ordered sets. Annals of Mathematics **51**(1), 161–166 (1950)
12. Gaudel, M.C.: Testing can be formal, too! In: 6th Int. Joint Conf. CAAP/FASE, Theory and Practice of Software Development, TAPSOFT'95, LNCS 915, pp. 82–96. Springer (1995)
13. Grieskamp, W., Kicillof, N., Stobie, K., Braberman, V.: Model-based quality assurance of protocol documentation: tools and methodology. Software Testing, Verification and Reliability **21**(1), 55–71 (2011)
14. Hagenah, C., Muscholl, A.: Computing epsilon-free NFA from regular expressions in $O(n \cdot \log(n)^2)$ time. Informatique Théorique et Applications **34**(4), 257–278 (2000)
15. Hennie, F.C.: Fault-detecting experiments for sequential circuits. In: 5th Annual Symposium on Switching Circuit Theory and Logical Design, pp. 95–110. IEEE Computer Society (1964)
16. Henniger, O.: On test case generation from asynchronously communicating state machines. In: 10th Int. Workshop on Testing of Communicating Systems, IWTCS'97, pp. 255–271. Chapman & Hall (1997)
17. Hierons, R.M.: The complexity of asynchronous model based testing. Theoretical Computer Science **451**, 70–82 (2012)
18. Hierons, R.M.: Implementation relations for testing through asynchronous channels. The Computer Journal **56**(11), 1305–1319 (2013)
19. Hierons, R.M., Bogdanov, K., Bowen, J., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luettgen, G., Simons, A., Vilkomir, S., Woodward, M., Zedan, H.: Using formal specifications to support testing. ACM Computing Surveys **41**(2) (2009)
20. Hierons, R.M., Bowen, J., Harman, M. (eds.): Formal Methods and Testing, LNCS 4949. Springer (2008)
21. Hierons, R.M., Merayo, M.G., Núñez, M.: Testing from a stochastic timed system with a fault model. Journal of Logic and Algebraic Programming **78**(2), 98–115 (2009)
22. Hierons, R.M., Merayo, M.G., Núñez, M.: Passive testing with asynchronous communications. In: IFIP 33rd Int. Conf. on Formal Techniques for Distributed Systems, FMOODS/FORTE'13, LNCS 7892, pp. 99–113. Springer (2013)
23. Hierons, R.M., Merayo, M.G., Núñez, M.: Timed implementation relations for the distributed test architecture. Distributed Computing **27**(3), 181–201 (2014)
24. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley (2006)
25. Hromkovic, J., Seibert, S., Wilke, T.: Translating regular expressions into small ϵ-free nondeterministic finite automata. Journal of Computer Systems and Science **62**(4), 565–588 (2001)
26. Huo, J., Petrenko, A.: On testing partially specified IOTS through lossless queues. In: 16th Int. Conf. on Testing Communicating Systems, TestCom'04, LNCS 2978, pp. 76–94. Springer (2004)
27. Huo, J., Petrenko, A.: Transition covering tests for systems with queues. Software Testing, Verification and Reliability **19**(1), 55–83 (2009)
28. Jard, C., Jéron, T., Tanguy, L., Viho, C.: Remote testing can be as powerful as local testing. In: 19th Joint Int. Conf. on Protocol Specification, Testing, and Verification and Formal Description Techniques, FORTE/PSTV'99, pp. 25–40. Kluwer Academic Publishers (1999)
29. Krichen, M., Tripakis, S.: Conformance testing for real-time systems. Formal Methods in System Design **34**(3), 238–304 (2009)
30. Lalanne, F., Maag, S.: A formal data-centric approach for passive testing of communication protocols. IEEE/ACM Transactions on Networking **21**(3), 788–801 (2013)
31. Laurent, O.: Using formal methods and testability concepts in the avionics systems validation and verification (V&V) process. In: 3rd Int. Conf. on Software Testing, Verification, and Validation, ICST'10, pp. 1–10. IEEE Computer Society (2010)
32. Lee, D., Chen, D., Hao, R., Miller, R., Wu, J., Yin, X.: A formal approach for passive testing of protocol data portions. In: 10th IEEE Int. Conf. on Network Protocols, ICNP'02, pp. 122–131. IEEE Computer Society (2002)

33. Lee, D., Chen, D., Hao, R., Miller, R., Wu, J., Yin, X.: Network protocol system monitoring: a formal approach with passive testing. IEEE/ACM Transactions on Networking **14**, 424–437 (2006)
34. Lee, D., Netravali, A., Sabnani, K., Sugla, B., John, A.: Passive testing and applications to network management. In: 5th IEEE Int. Conf. on Network Protocols, ICNP'97, pp. 113–122. IEEE Computer Society (1997)
35. Leucker, M., Schallhart, C.: A brief account of runtime verification. Journal of Logic and Algebraic Programming **78**(5), 293–303 (2009)
36. Mammar, A., Cavalli, A.R., Jimenez, W., Mallouli, W., Montes de Oca, E.: Using testing techniques for vulnerability detection in C programs. In: 23rd Int. Conf. on Testing Software and Systems, ICTSS'11, LNCS 7019, pp. 80–96. Springer (2011)
37. Marinescu, R., Seceleanu, C., Guen, H.L., Pettersson, P.: A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs, *Advances in Computers*, vol. 98, chap. 3, pp. 89–140. Elsevier (2015)
38. Mealy, G.: A method for synthesizing sequential circuits. Bell System Techical Journal **34**, 1045–1079 (1955)
39. Merayo, M.G., Núñez, A.: Passive testing of communicating systems with timeouts. Information and Software Technology **64**, 19–35 (2015)
40. Merayo, M.G., Núñez, M., Hierons, R.M.: Testing timed systems modeled by stream X-machines. Software and Systems Modeling **10**(2), 201–217 (2011)
41. Merayo, M.G., Núñez, M., Rodríguez, I.: Formal testing from timed finite state machines. Computer Networks **52**(2), 432–460 (2008)
42. Morales, G., Maag, S., Cavalli, A.R., Mallouli, W., Montes de Oca, E., Wehbi, B.: Timed extended invariants for the passive testing of web services. In: 8th IEEE Int. Conf. on Web Services, ICWS'10, pp. 592–599. IEEE Computer Society (2010)
43. Mouttappa, P., Maag, S., Cavalli, A.R.: Using passive testing based on symbolic execution and slicing techniques: Application to the validation of communication protocols. Computer Networks **57**(15), 2992–3008 (2013)
44. Myers, G.J.: The Art of Software Testing, 2nd edn. John Wiley & Sons (2004)
45. Nguyen, H.N., Poizat, P., Zaïdi, F.: Online verification of value-passing choreographies through property-oriented passive testing. In: 14th Int. Symposium on High-Assurance Systems Engineering, HASE'12, pp. 106–113. IEEE Computer Society (2012)
46. Nguyen, H.N., Zaïdi, F., Cavalli, A.R.: A framework for distributed testing of timed composite systems. In: 21st Asia-Pacific Software Engineering Conference, APSEC'14, pp. 47–54. IEEE Computer Society (2014)
47. Noroozi, N., Khosravi, R., Mousavi, M.R., Willemse, T.A.C.: Synchrony and asynchrony in conformance testing. Software and Systems Modeling **14**(1), 149–172 (2015)
48. Shu, G., Lee, D.: Message confidentiality testing of security protocols - passive monitoring and active checking. In: 18th Int. Conf. on Testing Communicating Systems, TestCom'06, LNCS 3964, pp. 357–372. Springer (2006)
49. Simão, A., Petrenko, A.: Generating asynchronous test cases from test purposes. Information and Software Technology **53**(11), 1252–1262 (2011)
50. Springintveld, J., Vaandrager, F., D'Argenio, P.: Testing timed automata. Theoretical Computer Science **254**(1-2), 225–257 (2001). Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997