Formal Testing from Timed Finite State Machines*

Mercedes G. Merayo[a], Manuel Núñez[a] and Ismael Rodríguez[a]

[a] Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, E-28040 Madrid, Spain
e-mail: mgmerayo@fdi.ucm.es,mn@sip.ucm.es,isrodrig@sip.ucm.es

In this paper we present a formal methodology to test both the functional and temporal behaviors in systems where temporal aspects are critical. We extend the classical finite state machines model with features to represent timed systems. Our formalism allows three different ways to express the timing requirements of systems. Specifically, we consider that time requirements can be expressed either by means of fix time values, by using random variables, or by considering time intervals. Different implementation relations, depending on both the interpretation of time and on the non-determinism appearing in systems, are presented and related. We also study how test cases are defined and applied to implementations. Test derivation algorithms, producing sound and complete test suites, are also presented. That is, by deriving these test suites we relate the different notions of passing tests and the different implementation relations. In other words, for a given correctness criterion, a system represents an appropriate implementation of a given model if and only if the system successfully passes all the test belonging to the derived test suite.

Keywords: Conformance Testing; Formal Methods; Timed Systems.

1. Introduction

The scale and heterogeneity of current systems make impossible for developers to have an overall view of them. Thus, it is difficult to foresee those errors that are either critical or more probable. In this context, *formal testing techniques* provide systematic procedures to check implementations in such a way that the coverage of critical parts/aspects of the system under test depends less on the intuition of the tester. In this line, they allow to test the correctness of a system with respect to a specification. Formal testing originally targeted the functional behavior of systems, such as determining whether the tested system can, on the one hand, perform certain actions and, on the other hand, does not perform some unexpected ones. While the relevant aspects of some systems only concern *what* they do, in some other systems it is equally relevant *how* they do what they do. Thus, after the initial consolidation stage, formal testing techniques started also to deal with *non-functional* properties such as the probability of an event to happen, the time that it takes to perform a certain action, or the time when a certain action happens. The work on formal testing applied to timed systems has attracted a lot of attention during the last years. In fact, there are already several proposals for timed testing (e.g. [22,7,17,31,28,10,24,12,9,20,19,3]). In these papers, time is considered to be *deterministic*, that is, time requirements follow the form "after/before t time units..." In fact, in most of the cases time is introduced by means of clocks following [1]. Even though the inclusion of time allows to give a more precise description of the system to be implemented, there are frequent situations that cannot be accurately described by using this notion of deterministic time. For example, in order to express that a message will arrive at any point of time belonging to the interval $[0, 1]$ we will need, in general, infinite transitions, one for each possible value belonging to the interval. In this case, it would be more appropriate to use time intervals to describe the system. Let us consider now that we have to simulate the performance of a petrol station. Since cars arrive in

*Research partially supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01) and the Marie Curie project TAROT (MRTN-CT-2003-505121).

such stations by following a Poisson distribution, we would need again to use an infinite number of transitions. Moreover, if we have to use a time interval we would be very imprecise since all that we could say is that the next car will arrive in the interval $[0, \infty)$. Thus, it would be very useful to have a mechanism allowing to express that a time constraint is given by using a random variable that follows a precise probability distribution function.

In this paper we study formal testing methodologies where the temporal behavior of systems is taken into account. In order to present our contribution, we will use a simple extension of the classical concept of *Finite State Machine*. Intuitively, transitions in finite state machines indicate that if the machine is in a state s and receives an input i then it will produce an output o and it will change its state to s'. An appropriate notation for such a transition could be $s \xrightarrow{i/o} s'$. If we consider a timed extension of finite state machines, transitions as $s \xrightarrow{i/o}_d s'$ indicate that the time between receiving the input i and returning the output o is given by d, where d belongs to a certain time domain. Even though we have chosen finite state machines, because they are widely used in the formal testing community, our results can be straightforwardly adapted to deal with (input-output) labelled transition systems; the extension of our results to deal with (timed) automata is more cumbersome, but not difficult taking as basis [28].

We consider three different domains to express temporal requirements: Time given by *fix values*, by *random variables*, and by *time intervals*. A transition such as $s \xrightarrow{i/o}_t s'$ indicates that if the machine is in state s and receives the input i, it will perform the output o and reach the state s' after t time units. A transition as $s \xrightarrow{i/o}_\xi s'$ indicates that if the machine is in state s and receives the input i, it will perform the output o and reach the state s' after a certain time t with probability $F_\xi(t)$, where F_ξ is the probability distribution function associated with ξ. Finally, $s \xrightarrow{i/o}_{[t_1,t_2]} s'$ means that if the machine is in state s and receives the input i, it will perform the output o and reach the state s', and it will take a time greater than or equal to t_1 but smaller than or equal to t_2. Even though our methodology allows three very different ways for representing time requirements, random variables and time intervals present a more complex situation than fix time values. Thus, we need to treat them separately, although following a common line. Specifically, due to the fact that we work under the assumption of a black-box testing framework, testers cannot compare in a direct way timed requirements of the *real* implementation with those established in the model (either random variables or time intervals). The idea is that we can *see* the random variable (or the time interval) defining a given transition in the model, but we cannot do the same with the corresponding transition of the implementation, since we do not have access to it. Thus, in contrast with approaches considering fix time values, to perform a transition of the implementation once does not allow us to obtain all the information about its temporal behavior. In order to overcome this problem, we have to perform the same transition to collect different time values. So, we consider a set of observations collected by means of the interaction with the implementation and establish different levels of temporal agreement with respect to the (accessible) values appearing in the formal model. We think that this additional complication is the main reason why there is almost no work on testing timed systems where time is not given by means of fix time values. In fact, as far as we know, [25] represents the only proposal presenting a formal testing methodology to test stochastic time systems that can be described by means of finite state machines. Also, [2,21] present testing frameworks for stochastic systems but their approaches are not related to ours since they take as starting point the classical *de Nicola & Hennessy* [8,15] methodology. Let us note that this additional machinery is not necessary in the case of fix time values. This is so because if we observe that the performance of a transition in the implementation takes t time units, we know that any further performances will always take the same time t, while this is not the case if the time requirement is described either by using a

random variable or a time interval.

We study *conformance testing* relations to relate implementations with formal models. First, we will introduce an implementation relation where time is not considered. The idea is that the implementation I does not *invent* anything for those inputs that are *specified* in the model. In order to cope with time, we do not take into account only that a system may perform a given action but we also record the amount of time that the system needs to do so. We propose several timed conformance relations according to the interpretation of *good* implementation and the different time domains we consider. Time aspects add extra complexity to the task of defining these relations. For example, even though an implementation I had the same traces as a formal model S, we should not consider that I conforms to S if I is always *slower* than S. Moreover, it can be the case that a system performs the same sequence of actions for different times. These facts motivate the definition of several conformance relations. For example, it can be said that an implementation conforms to a formal model if the implementation is always *faster*, or if the implementation is at least as *fast* as the worst case of the model. In a first approach, and in order to improve readability, we will restrict ourselves to *observable non-deterministic* systems. We say that a machine is observable non-deterministic, or simply observable, if it does not have two transitions such as $s \xrightarrow{i/o}_{d_1} s_1$ and $s \xrightarrow{i/o}_{d_2} s_2$. However, we allow transitions such as $s \xrightarrow{i/o_1}_{d_1} s_1$ and $s \xrightarrow{i/o_2}_{d_1} s_2$, as far as $o_1 \neq o_2$. Thus, we can still specify *partially* non-deterministic behaviors. In the last part of the paper we suppress this condition and extend our work for dealing with fully non-deterministic systems.

With respect to the application of tests to implementations, the above mentioned non-deterministic temporal behavior requires that tests work in a specific manner. For example, if we apply a test and we observe that the implementation takes less time than the one required by the formal model, then this single application of the test allows us to know that the implementation *may* be faster than the model, but not that it *must* be so.

We would like to conclude this introduction by pointing out one of the limitations of black box testing. Essentially, if the tester does not make any assumption on the system under test, then the verdict of the testing process will be, most of the times, inconclusive. In other words, the tester will not be able to ensure that the system does not present an error. Let us consider the following *specification*:

```
x in {0,1}
while true do begin
  read(x); write(x)
end
```

Let us suppose that we have to test whether a given program conforms to this specification and we do not have access to the code of the program. In order to test the program, we have to provide values, check the screen, and if they coincide then the implementation is correct. However, if we have no information about the internal structure of the program, how many times do we give values? Three times is enough? 100? In fact, we can have the following *implementation*:

```
Program simple;
  begin
     read(x); write(x);
     read(x); write(x);
     read(x); write(x);
     read(x); write(2)
  end.
```

In order to overcome this problem, some formal testing approaches assume that some information about the system under test is available. For example, it is very common to consider that the implementation has a known number of states and that it is *somehow* deterministic. In our example, if we assume that our black-box implementation has, for instance, one state (let us note that our program `simple` has more than one state) and that it is deterministic then there exist testing methods to ensure that the implementation is indeed either correct or faulty. In this paper we do not assume this additional information. Thus, in

general, we will not be able to provide a conclusive verdict. However, our methodology allows to ensure correctness *in the limit* of any implementation under test (that is, without assuming additional hypothesis). In this running example, we would be able to provide verdicts such as "the program works as expected if we provide n values or less." If n tends to infinite then, in the limit, we can prove the correctness of the black-box.

Some of the results appearing in this paper have appeared in [25,26]. The main contribution of this paper with respect to this previous work is to present a unified framework containing the formalisms and results appearing in these papers. In addition, this paper includes the work on time intervals (this was not treated in previous papers) and contains proofs of the main results as well as additional explanations and examples.

The rest of the paper is organized as follows. In Section 2 we introduce additional notation used along the paper. In Section 3 we present our timed formalism. In Section 4, our timed conformance relations for each of the time domains are considered. In Section 5 we show how tests are defined and describe how to apply them to implementations. In Section 6 we present an algorithm to derive sound and complete test suites with respect to the implementation relations presented in Section 4. In Section 7 we present an extension of our work considering non-deterministic behaviors of systems. We devote Section 8 to related work. In Section 9 we give some concluding remarks. Finally, in Section 10 we present an appendix where Pearson's χ^2 hypothesis contrast is described.

2. Preliminaries

Along this paper, we consider that time values belong to a generic domain $\mathcal{T}\mathtt{ime}$. Most concepts will be parameterized with respect to this domain. However, some of the notions and definitions will depend on the concrete instance of the generic time domain. Specifically, we will consider three different possibilities to represent time: Time values, stochastic time, and time intervals. Regarding time values, we will use $\mathbb{R}_+$ as time domain. We need to introduce notation, related to stochastic time and time intervals, that we will use during the rest of the paper.

Definition 1 We say that $\hat{a} = [a_1, a_2]$ is a *time interval* if $a_1 \in \mathbb{R}_+$, $a_2 \in \mathbb{R}_+ \cup \{\infty\}$, and $a_1 \leq a_2$. We assume that for all $r \in \mathbb{R}_+$ we have $r < \infty$ and $r + \infty = \infty$. We consider that $\mathcal{I}_{\mathbb{R}_+}$ denotes the set of time intervals. Let $\hat{a} = [a_1, a_2]$ and $\hat{b} = [b_1, b_2]$ be time intervals. We write

- $\hat{a} \subseteq \hat{b}$ if we have both $b_1 \leq a_1$ and $a_2 \leq b_2$;
- $\hat{a} \preceq \hat{b}$ if we have both $a_1 \leq b_1$ and $a_2 \leq b_2$;
- $\hat{a} \ll \hat{b}$ if we have $a_2 \leq b_1$.

In addition, we introduce the following notation

- $\hat{a} + \hat{b}$ denotes the interval $[a_1 + b_1, a_2 + b_2]$;
- $\pi_i(\hat{a})$, for $i \in \{1, 2\}$, denotes the value a_i.

□

Time intervals will be used to express time constraints associated with the performance of actions. The idea is that if we associate a time interval $[t_1, t_2] \in \mathcal{I}_{\mathbb{R}_+}$ with a task we want to indicate that the associated task should take at least t_1 time units and at most t_2 time units to be performed. Let us note that in the case of $[t_1, \infty]$ and $[0, \infty]$ we are abusing the notation since these intervals are half-closed intervals, that is, they represent the intervals $[t_1, \infty)$ and $[0, \infty)$, respectively.

We use random variables to model *stochastic time*. Thus, we need to introduce some basic concepts on random variables.

Definition 2 We denote the set of random variables by $\mathcal{V}$ ($\xi, \psi, \ldots$ to range over $\mathcal{V}$). We will consider that the sample space (that is, the domain of random variables) is $\mathbb{R}_+$. The reason for this restriction is that random variables will always be associated with time distributions, so they cannot take a negative value.

Let $\xi \in \mathcal{V}$ be a random variable. We define the *probability distribution function* associated to ξ as the function $F_\xi : \mathbb{R}_+ \longrightarrow [0, 1]$ such that $F_\xi(x) = P(\xi \leq x)$, where $P(\xi \leq x)$ is the probability that ξ assumes values less than or equal to x.

Given two random variables $\xi, \xi' \in \mathcal{V}$ we consider that $\xi + \xi'$ denotes a random variable distributed as the addition of the two random variables ξ and ξ'. Let F_ξ and $F_{\xi'}$ be the probability distribution functions of ξ and ξ', respectively. We write $\xi = \xi'$ if for all $x \in \mathbb{R}$ we have $F_\xi(x) = F_{\xi'}(x)$.

We will call *sample* to any multiset of positive real numbers. We denote the set of multisets in $\mathbb{R}^+$ by $\wp(\mathbb{R}^+)$. Let ξ be a random variable and J be a sample. We denote the *confidence* of ξ on J by $\gamma(\xi, J)$. □

In the previous definition, a sample simply denotes a multiset of observed values. For example, let us toss a coin one hundred times. A sample consists of the different values that we observe, that is, n heads and $100 - n$ tails. In our setting, samples will be associated with time values that implementations need to perform sequences of actions. We have that $\gamma(\xi, J)$ takes values in the interval $[0, 1]$. Intuitively, bigger values of $\gamma(\xi, J)$ indicate that the observed sample J is more likely to be produced by the random variable ξ. That is, this function decides how *similar* the probability distribution function generated by J and the one corresponding to the random variable ξ are. Intuitively, if we are testing whether a coin is fair, we toss it one hundred times, and we observe 89 heads and 11 tails, we will conclude that, with a high probability, the coin is not fair. In the appendix of this paper we show how confidence is formally defined.

In order to avoid side-effects, we will always assume that all the random variables appearing in the definition of a TFSM are independent. Let us note that this condition does not restrict the distributions to be used. In particular, there can be random variables identically distributed even though they are independent.

3. A timed extension of the FSM model

In this section we introduce our timed extensions of the classical finite state machine model. The main difference with respect to usual FSMs consists in the addition of *time* to indicate the lapse between offering an input and receiving an output.

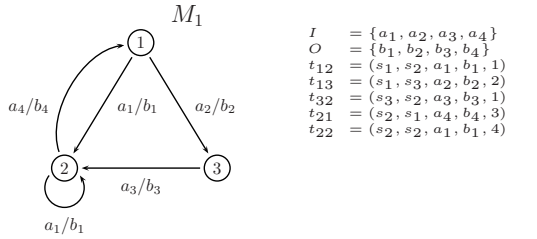


Figure 1. Example of TFSM: Fix time values.

3.1. Basic concepts on TFSMs

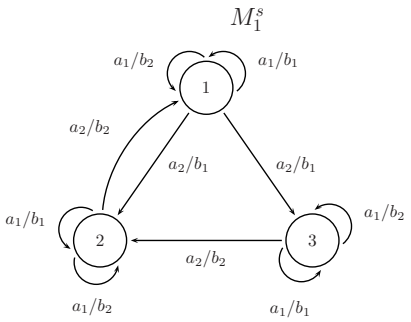
Definition 3 Let $\mathcal{T}\mathtt{ime}$ be the time domain. A *Timed Finite State Machine*, in the following TFSM, is a tuple $M = (S, I, O, Tr, s_{in})$ where S is a finite set of states, I is the set of input actions, O is the set of output actions, Tr is the set of transitions, and s_{in} is the initial state.

A transition belonging to Tr is a tuple (s, s', i, o, d) where $s, s' \in S$ are the initial and final states of the transition, $i \in I$ and $o \in O$ are the input and output actions, respectively, and $d \in \mathcal{T}\mathtt{ime}$ denotes the time that the transition needs to be completed.

We say that M is *input-enabled* if for all state $s \in S$ and input $i \in I$, there exist $s' \in S$, $o \in O$, and $d \in \mathcal{T}\mathtt{ime}$ such that $(s, s', i, o, d) \in Tr$. We say that M is *observable non-deterministic*, or simply *observable*, if there do not exist two different transitions (s, s_1, i, o, d_1) and (s, s_2, i, o, d_2) belonging to Tr. □

Intuitively, a transition (s, s', i, o, d) indicates that if the machine is in state s and receives the input i then, after the time defined by d, the machine emits the output o and moves to s'. Depending on the notion of time that we are using, d may be a time value, a random variable, or a time interval. Specifically, if we are considering *fix time values* then $d \in \mathbb{R}_+$ is a non-negative real number, if we are considering *stochastic time* then $d \in \mathcal{V}$ is a random variable, and if we are considering *time intervals* then $d \in \mathcal{I}_{\mathbb{R}_+}$.

Example 1 In Figure 1 we present a TFSM. In this case, the time associated with each action is a non-negative real value. Let us suppose that



$$F_1(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ \frac{x}{3} & \text{if } 0 < x < 3 \\ 1 & \text{if } x \geq 3 \end{cases}$$

$$F_2(x) = \begin{cases} 0 & \text{if } x < 4 \\ 1 & \text{if } x \geq 4 \end{cases}$$

$$F_3(x) = \begin{cases} 1 - e^{-3 \cdot x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

$I = \{a_1, a_2\}, O = \{b_1, b_2\}$
$t_{111} = (s_1, s_1, a_1, b_1, \xi_{11})$
$t_{112} = (s_1, s_1, a_1, b_2, \xi_{12})$
$t_{12} = (s_1, s_2, a_2, b_1, \xi_{13})$
$t_{13} = (s_1, s_3, a_2, b_1, \xi_{14})$
$t_{21} = (s_2, s_1, a_2, b_2, \xi_{15})$
$t_{221} = (s_2, s_2, a_1, b_1, \xi_{21})$
$t_{222} = (s_2, s_2, a_1, b_2, \xi_{22})$
$t_{32} = (s_3, s_2, a_2, b_2, \xi_{23})$
$t_{331} = (s_3, s_3, a_1, b_1, \xi_{31})$
$t_{332} = (s_3, s_3, a_1, b_2, \xi_{32})$

where ξ_{ij} are independent random variables distributed by following F_i

Figure 2. Example of TFSM: Stochastic time.

the initial state of M_1 is the state labelled by 1. Then, the transition t_{12} can be performed and it will take time 1.

Let us consider the machine depicted in Figure 2. In this case time values are stochastic, that is, the time function assigns a random variable to each transition. Depending on the system that we are modelling, these random variables will follow a specific probability distribution function. In this example we show three possible, often used, probability distribution functions. For instance, we may consider that for all $1 \leq j \leq 5$ we have that the ξ_{1j} random variables are *uniformly distributed* in the interval $[0, 3]$, that is, they have F_1 as associated probability distribution function. Uniform distributions allow us to keep compatibility with time intervals in (non-stochastic) timed models in the sense that the same *weight* is assigned to all the times in the interval. We may consider that for all $1 \leq j \leq 3$ we have that the ξ_{2j} random variables follow a Dirac distribution in 4, that is, they have F_2 as associated probability distribution function. A Dirac distribution concentrates all the probability in a single point, that is, a Dirac distribution in n gives probability 1 to n and probability 0 to the rest of values. In timed terms, the idea is that the corresponding delay will be equal to n time units. Dirac distributions allow us to simulate deterministic delays appearing in timed models. Finally, ξ_{31} and ξ_{32} are *exponentially* distributed with parameter 3, that is, they have F_3 as associated probability distribution function.

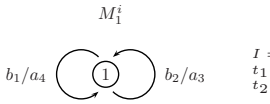


Figure 3. Example of TFSM: Time intervals.

For instance, let us consider the transition t_{12}. Intuitively, if the machine is in state 1 and receives the input a_2 then it will produce the output b_1 after a time given by ξ_{13}. Since ξ_{13} is uniformly distributed in $[0, 3]$ we have that, for example, this time will be less than 1 time unit with probability $\frac{1}{3}$, it will be less than 1.5 time units with probability $\frac{1}{2}$, and so on. Finally, once 3 time units have passed we know that the output b_1 has been performed (that is, we have probability 1).

In Figure 3 we present a TFSM where requirements on the time consumed by actions are given by means of intervals. The initial state of M_1^i is the only state in the machine, labelled by 1. Then, the transition t_1 will take a time belonging to $[1, 3]$, that is, a time greater than or equal to 1 and less than or equal to 3. □

Next, we introduce the notion of *trace*. As usual, a trace is a sequence of input/output pairs. In addition, we have to record the time that the trace needs to be performed. An *evolution* is a trace starting at the initial state of the machine.

Definition 4 Let $M = (S, I, O, Tr, s_{in})$ be a TFSM. We say that $(s, s', (i_1/o_1, \ldots, i_r/o_r), d)$ is a *timed trace*, or simply *trace*, of M if there exist (s, s_1, i_1, o_1, d_1),..., $(s_{r-1}, s', i_r, o_r, d_r) \in Tr$, such that $d = \sum d_i$.

We say that $(i_1/o_1, \ldots, i_r/o_r)$ is a *non-timed evolution*, or simply *evolution*, of M if we have that $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), d)$ is a trace of M. We denote by $\texttt{NTEvol}(M)$ the set of non-timed evolutions of M.

We say that the pair $((i_1/o_1, \ldots, i_r/o_r), d)$ is a *timed evolution* of M if we have that $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), d)$ is a trace of M. We denote by $\texttt{TEvol}(M)$ the set of timed evolutions of M.

Let $n \geq 1$, $i_1, \ldots, i_n \in I$, and $o_1, \ldots, o_{n-1} \in O$. We define the *set of outputs after* the sequence $i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n$ as

$$\texttt{setout}(M, i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n) = \left\{ o \in O \;\middle|\; \begin{array}{l} \exists s, s' \in S, d, d' \in \mathcal{T}\texttt{ime} : \\ \quad (s_{in}, s, (i_1/o_1, \ldots, i_{n-1}/o_{n-1}), d) \\ \qquad \text{timed trace of } M \\ \quad \wedge\, (s, s', i_n, o, d') \in Tr \end{array} \right\}$$

□

Traces are sequences of transitions. The time associated with a trace is computed from the corresponding to each transition belonging to the sequence. In fact, this time is obtained by adding the time values associated with each of the transitions conforming the trace. Let us remark that the actual representation of time can be abstracted. Specifically, when defining the time d associated with a trace, we can give a generic definition since addition is defined for time fix values (as usual in real numbers) as well as for time intervals and random variables (see Definitions 1 and 2). The setout function will be used in the forthcoming Lemma 1 when we provide an alternative characterization of our non-timed implementation relation. Intuitively, $\texttt{setout}(M, i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n)$ computes those outputs that can conform a transition together with i_n after the machine performs the sequence $i_1/o_1, \ldots, i_{n-1}/o_{n-1}$. Let us remark that if the machine cannot perform the sequence then setout will return the empty set.

Example 2 Let us consider again the TFSM depicted in Figure 1 and its transitions t_{13}, t_{32}, and t_{21}. We can build the trace $(s_1, s_1, (a_2/b_2, a_3/b_3, a_4/b_4), 6)$ based on these transitions. This trace represents that from state 1 the machine can accept the sequence of inputs (a_2, a_3, a_4) and it will emit the sequence of outputs (b_2, b_3, b_4) after 6 time units pass. □

3.2. Testing hypotheses

In this section we describe the characteristics and restrictions that we will consider along the paper regarding implementations under test and formal models.

We consider that models are given by TFSMs. Regarding implementations, we also assume that they are also given by means of TFSMs, more precisely, and following the classical assumption in formal testing, we suppose that there exists a model of the implementation that can be represented by an TFSM. Besides, we assume that input actions are always enabled in any state of the implementation, that is, implementations are input-enabled according to Definition 3. This is a usual condition to assure that the implementation will react (somehow) to any input.

By now, and in order to simplify the presentation, we will consider that both specifications and implementations are given by observable TFSMs (see Definition 3). Let us note that even restricting to this kind of machines we may still have two transitions (s, s_1, i, o_1, d_1) and (s, s_2, i, o_2, d_2), as far as $o_1 \neq o_2$. Thus, we allow some degree of non-determinism in this first approach. In Section 7 we will show how our framework can be extended to deal with fully non-deterministic behaviors.

Regarding actions, we suppose that when checking the validity of an implementation with respect to a model, they have the same sets of

inputs and outputs. Let us remark that this condition can be always easily achieved by considering the union of the corresponding sets and assuming that the sets of inputs and outputs are given by these new sets. Formally, if we have two TFSMs $M_1 = (S_1, I_1, O_1, Tr_1, s_{in}^1)$ and $M_2 = (S_2, I_2, O_2, Tr_2, s_{in}^2)$ we can assume that these two machines are in fact given by $M_1 = (S_1, I_1 \cup I_2, O_1 \cup O_2, Tr_1, s_{in}^1)$ and $M_2 = (S_2, I_1 \cup I_2, O_1 \cup O_2, Tr_2, s_{in}^2)$, respectively. In this paper we do not explicitly consider a `null` action. Some approaches based on finite state machines allow inputs and outputs not to be always paired. In this case, they assume the existence of special symbols denoting the omission of the corresponding input or output. We deal with these special symbols in the same way as we do with any other input or output. For example, if we are specifying a machine such that there are input actions that are not followed by an output then we will use transitions such as $s \xrightarrow{i/\texttt{null}}_d s'$ and we will consider that `null` belongs to the set of outputs.

4. Implementation relations

In this section we introduce our implementation relations. This kind of relations are a useful theoretical tool to relate implementations and specifications. The idea is that an implementation is related to a formal model iff the implementation is *correct* with respect to the model. It is the meaning of correctness what produces that there does not exist a unique way to define an implementation relation. In the case of timed systems, what is a good implementation is even less precise. For example, one may consider that I is a good implementation of S if I takes the same time to perform its tasks as S while another could consider that the implementation has to be always/sometimes faster. Thus, for each of these considerations, we will define a different implementation relation. One may wonder whether it is necessary to have more than just one implementation relation, as it is usually the case when dealing with systems where time is not considered. The problem is that there is not a good argument to consider that one implementation relation is *better*, in the sense that it more accurately characterizes what a good implementation is, than another. Thus, in this paper we will give different relations, for each possible choice of the time domain. It will be the user of our framework who will have to decide which implementation relations better suites her idea of what a good implementation is. In the same way, in the forthcoming Section 5 we will not define a unique way of what to successfully pass a test means. For example, one may consider that in order to successfully pass a test, the time that it took to apply the test must be always smaller than the one expected by the test while another tester might think that in order to consider that the application of a test was successful, the used time had to be equal to the one expected by the test. Again, it will be the tester who will decide which notion she wants to use.

Even though, according to its timed behavior, there are different possibilities to consider what a good implementation is, there is an agreement on correctness if we consider only functional behavior, that is, abstracting time. Regarding the performance of usual inputs and outputs, an implementation I should not *invent* behaviors when provided with inputs specified in the formal model S. This means that the *relevant* evolutions of I must be contained in those corresponding to S. Thus, all our implementation relations follow the same pattern: I *conforms* to S if for all possible evolution of S the outputs that the implementation I may perform after a given input are a subset of those of S. This pattern is borrowed from `ioco` [29,30]. However, we do not consider *quiescent* states, that is, states where no external outputs are available, since finite state machines, and their variants, have a strict alternation between inputs and outputs. Thus, a notion of quiescence does not apply to this framework.

It is worth to mention that implementation relations are usually *related* to some notion of passing tests. Thus, most formal testing theories have results such as "a system is a good implementation with respect to a given implementation relation iff the implementation successfully passes a test suite extracted from the formal model." In our framework, as we will show in Sections 6 and 7.3, we have such results for the implemen-

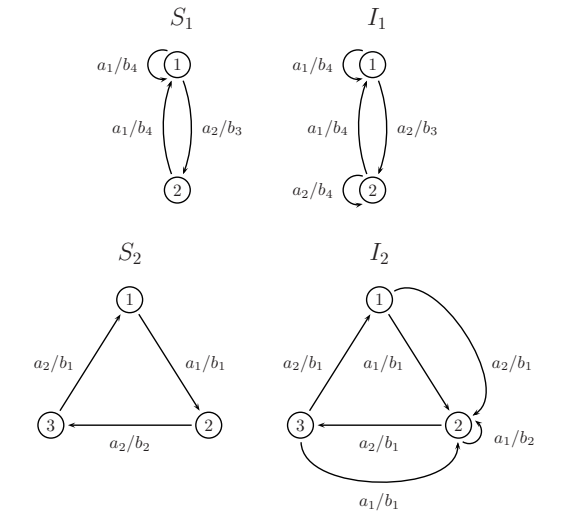


Figure 4. Examples of (non-)timely conformance.

tation relations defined in this paper.

First, we introduce an implementation relation where time is not considered. This relation, firstly introduced in [24], will be the basis to define the rest of notions.

Definition 5 Let S and I be two TFSMs. We say that I *non-timely conforms* to S, denoted by I $\mathtt{conf}_{nt}$ S, if for all $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r) \in \mathtt{NTEvol}(S)$, with $r \geq 1$, we have that

$$e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r) \in \mathtt{NTEvol}(I)$$
$$\Downarrow$$
$$e' \in \mathtt{NTEvol}(S)$$

□

As we said before, the idea underlying the definition of the non-timely conformance relation $\mathtt{conf}_{nt}$ is that the implementation does not *invent* anything for those inputs that are *specified* in the formal model. Let us note that if the model had also the property of input-enabled then we could remove the condition "for all $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r) \in \mathtt{NTEvol}(S)$, with $r \geq 1$."

Example 3 Let us consider the systems S_1 and I_1 depicted in Figure 4 (time information has not been included since it is not relevant for this example). We have I_1 $\mathtt{conf}_{nt}$ S_1. Let us note that the non-timed evolutions of I_1 having as prefix the sequence $(a_2/b_3, a_2/b_4)$ are not checked because the specification S_1 cannot perform those evolutions.

Let us now consider S_2 and I_2 as depicted in the same figure. We have that I_2 does not conform to S_2. For example, S_2 may perform the non-timed evolution $e = (a_1/b_1, a_2/b_2)$, I_2 has the non-timed evolution $e' = (a_1/b_1, a_2/b_1)$, but e' does not belong to the set of non-timed evolutions of S_2. Let us note that the sequence e' has to be checked since e and e' have the same prefix $a_1/b_1, a_2$. □

Since the previously introduced implementation relation is the one that we will use to define the rest of implementation relations given in this paper, it can be useful to have an alternative definition. The following result states that the $\mathtt{conf}_{nt}$ relation can be alternatively defined by considering those outputs that can be performed after some input/output sequences are performed. These sequences are selected from those that the formal model S can perform, that is, we consider only those sequences e such that $\mathtt{setout}(S, e) \neq \emptyset$.

Lemma 1 Let S and I be two TFSMs. We have that I $\mathtt{conf}_{nt}$ S iff for all sequence of input/outputs $i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n$ such that $\mathtt{setout}(S, i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n) \neq \emptyset$ we have

$$\mathtt{setout}(I, i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n)$$
$$\subseteq$$
$$\mathtt{setout}(S, i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n)$$

□

Next we introduce our first timed implementation relations distinguishing the cases where temporal requirements are specified by a time value, a random variable, and a time interval. In addition to the non-timed conformance of the implementation, we require some time conditions to hold. As we explained before, the different considerations of time produce that there is not a unique way to define an implementation relation.

4.1. Fix time implementation relations

We present two relations for the case when time is expressed by fix values. In the $\texttt{conf}_a$ relation (conforms *always*) we consider that for all timed evolution (e,t) of the implementation, if e is a non-timed evolution of the model S then (e,t) is also a timed evolution of S. With this relation we express that the implementation mimics the timed behavior of the formal model. In the $\texttt{conf}_b$ relation (conforms in the *best* case) the implementation is forced, for each timed evolution fulfilling the previous conditions, to be faster than the timed evolution in the model. This implementation relation expresses that the implementation is faster than the model. Let us remark that since we are considering observable machines, there is at most one evolution fulfilling the previous conditions. In Section 7 we will see that the removal of this restriction gives rise to multiple alternative notions because, for each e, there can be several time values t such that (e,t) is a timed evolution.

Definition 6 Let S and I be two TFSMs. We define the following implementation relations:

- *(always)* $I\ \texttt{conf}_a\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time value $t \in \mathbb{R}_+$

$$(e,t) \in \texttt{TEvol}(I) \Longrightarrow (e,t) \in \texttt{TEvol}(S)$$

- *(best)* $I\ \texttt{conf}_b\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time value $t \in \mathbb{R}_+$

$$\begin{array}{c}(e,t) \in \texttt{TEvol}(I)\\ \Downarrow \\ \exists\, t' \in \mathbb{R}_+ : ((e,t') \in \texttt{TEvol}(S)\ \wedge\ t \leq t')\end{array}$$

□

Let us remark that since $\texttt{conf}_{nt}$ is a requirement for $\texttt{conf}_a$ and $\texttt{conf}_b$, we can restrict ourselves to study time values associated with evolutions common to both S and I. This is due to the fact that the implementation cannot show unexpected evolutions. That is the reason why we consider $\texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$. Something similar happens in all the timed implementation relations presented along this paper.

4.2. Stochastic implementation relations

Next we introduce our first implementation relation for finite state machines where time requirements are defined by using random variables.

Definition 7 Let I and S be two TFSMs. We say that I *stochastically conforms* to S, denoted by $I\ \texttt{conf}_s\ S$, if $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all random variable $\xi \in \mathcal{V}$

$$\begin{array}{c}(e,\xi) \in \texttt{TEvol}(I)\\ \Downarrow \\ \exists\, \xi' \in \mathcal{V}\ : ((e,\xi') \in \texttt{TEvol}(S)\ \wedge\ \xi = \xi')\end{array}$$

□

In addition to requiring the notion of *non-timely* conformance, we have to ask for some conditions on the corresponding random variables. Thus, $I\ \texttt{conf}_s\ S$ also requires that all timed evolutions of S that can be performed by the implementation must have identically distributed random variables. Even though this is a very reasonable notion of conformance, the fact that we assume a black-box testing framework disallows us to check whether the corresponding random variables are identically distributed. In fact, we would need an infinite number of observations from a random variable of the implementation (with an unknown distribution) to assure that this random variable is distributed as another random variable from the formal model (with a known distribution). Thus, we have to give more *realistic* implementation relations based on a finite set of observations. We will present other implementation relations that are less *accurate* but are *checkable*. We only need to suppose that we can actually record the time that the implementation needs to perform a given sequence.

Definition 8 Let I be a TFSM. We say that $((i_1/o_1,\ldots,i_n/o_n),t)$ is the *observed timed execution* of I, or simply *timed execution*, if the observation of I shows that the sequence $(i_1/o_1,\ldots,i_n/o_n)$ is performed in time t.

Let Φ be a set of input/output sequences and $H = \{\!\{(e_1,t_1),\ldots,(e_n,t_n)\}\!\}$ be a multiset of timed executions. We say that the function

$\texttt{Sampling}_{(H,\Phi)} : \Phi \longrightarrow \wp(\mathbb{R}^+)$ is a *sampling application* of H for Φ if for all $e \in \Phi$ we have $\texttt{Sampling}_{(H,\Phi)}(e) = \{\!|t \mid (e,t) \in H|\!\}$. □

Timed executions are input/output sequences together with the time that it took to perform the sequence. In a certain sense, timed executions can be seen as *instances* of the evolutions that the implementation can perform. Regarding the definition of sampling applications, we just associate with each evolution the observed execution time values. These time values will be compared, by using a contrast hypothesis, with the random variable associated with the same evolution in the model.

Definition 9 Let I and S be two TFSMs, H be a multiset of timed executions of I, $0 \leq \alpha \leq 1$, and $\Phi = \{e \mid \exists\, t : (e,t) \in H\} \cap \texttt{NTEvol}(S)$. We say that I $(\alpha, H)-$*stochastically conforms* to S, denoted by $I\, \texttt{conf}_s^{(\alpha,H)}\, S$, if $I\, \texttt{conf}_{nt}\, S$ and for all evolution $e \in \Phi$ we have that $(e,\xi) \in \texttt{TEvol}(S)$ implies $\gamma(\xi, \texttt{Sampling}_{(H,\Phi)}(e)) > \alpha$. □

The idea underlying the new relation is that the implementation must conform to the specification in the usual way (that is, $I\, \texttt{conf}_{nt}\, S$). Besides, for all evolution of the implementation that can be performed by S and that has been observed (i.e. it is included in H), the observed execution time values *fit* the random variables indicated by S. This notion of *fitting* is given by the function γ that is formally defined in the appendix. Let us remark that the set H plays a fundamental role in the previous definition and in the forthcoming implementation relations dealing either with stochastic time or with time intervals. The idea is that the bigger the set H is, the more confident we are that the hypothesis contrast (that is, the application of γ) returns a result that reflects the real situation of the implementation with respect to the model. Obviously, if H is too small then the knowledge that we have about the temporal behavior of the implementation is very limited and it is more likely that we reach a wrong conclusion. We will comment on this point again in Section 5.2 when we define how to test timed systems where time information is given by means of random variables.

A first direct result says that if we decrease the confidence level then we keep conformance.

Lemma 2 Let I and S be two TFSMs such that $I\, \texttt{conf}_s^{(\alpha_1,H)}\, S$. If $\alpha_2 < \alpha_1$ then we have $I\, \texttt{conf}_s^{(\alpha_2,H)}\, S$. □

The next result, whose proof is straightforward, says that if we have two samples sharing some properties then our conformance relation gives the same result for both of them.

Lemma 3 Let I and S be two TFSMs, H_1, H_2 be multisets of timed executions for I, and $b_i = \{\!|(e,t) \mid (e,t) \in H_i \wedge e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)|\!\}$, for $i = \{1,2\}$. If $b_1 = b_2$ then we have $I\, \texttt{conf}_s^{(\alpha,H_1)}\, S$ iff $I\, \texttt{conf}_s^{(\alpha,H_2)}\, S$. □

During the rest of this section we present different variations of the previous implementation relation. First, we define the concept of *shifting* a random variable with respect to its mean. For example, let us consider a random variable ξ following a Dirac distribution in 4 (see Example 1 for the formal definition). If we consider a new random variable ξ' following a Dirac distribution in 3, we say that ξ' represents a shift of ξ. Moreover, we also say that ξ and ξ' belong to the same family.

Definition 10 We say that ξ' is a *mean shift* of ξ with mean M', and we denote it by $\xi' = \texttt{MShift}(\xi, M')$, if ξ, ξ' belong to the same family and the mean of ξ', denoted by $\mu_{\xi'}$, is equal to M'.

Let I and S be two TFSMs, H be a multiset of timed executions of I, $0 \leq \alpha \leq 1$, and $\Phi = \{e \mid \exists\, t : (e,t) \in H\} \cap \texttt{NTEvol}(S)$. We say that I $(\alpha, H)-$*stochastically conforms* to S *with speed* π, denoted by $I \texttt{confm}_\pi^{(\alpha,H)} S$, if $I\, \texttt{conf}_{nt}\, S$ and for all $e \in \Phi$ we have that (e,ξ) $\texttt{TEvol}(S)$ implies $\gamma(\texttt{MShift}(\xi, \mu_\xi \cdot \pi), \texttt{Sampling}_{(H,\Phi)}(e)) > \alpha$. □

An interesting remark regarding this new relation is that when α is *small enough* and/or π is *close enough* to 1, it may happen that we have both $I\, \texttt{conf}_s^{(\alpha,H)}\, S$ and $I \texttt{confm}_\pi^{(\alpha,H)} S$. Nevertheless, it is enough to increase α, as far as $\pi \neq 1$,

so that we do not have both results simultaneously. Let us note that in the previous notion, a value of π greater than 1 indicates that the implementation is *slower*. This observation induces the following relation.

Definition 11 Let I and S be two TFSMs and H be a multiset of timed executions of I. We say that I is *generally faster* (respectively *generally slower*) than S for H if there exist $0 \leq \alpha \leq 1$ and $0 < \pi < 1$ (respectively $\pi > 1$) such that $I\,\mathtt{confm}_{\pi}^{(\alpha,H)} S$ but $I\,\mathtt{conf}_{s}^{(\alpha,H)}\ S$ does not hold. □

Given the fact that, in our framework, an implementation I could *fit better* a model S with higher or lower speed, it will be interesting to detect which variations of speed would make the implementation to fit better the model. Intuitively, the best variation will be the one allowing I to conform to S with a *higher* level of confidence α.

Definition 12 Let I and S be two TFSMs and H be a multiset of timed executions of I. Let us consider $0 \leq \alpha \leq 1$ and $\pi \in \mathbb{R}_+$ such that $I\,\mathtt{confm}_{\pi}^{(\alpha,H)} S$ and there do not exist $\alpha' > \alpha$ and $\pi' \in \mathbb{R}_+$ with $I\,\mathtt{confm}_{\pi'}^{(\alpha',H)} S$. Then, we say that π is a *relative speed* of I with respect to S for H. □

The concept of relative speed allows us to define another implementation relation which is more restrictive than those presented so far. Basically, the implementation must both $(\alpha, H)-$stochastically conform to S and have 1 as a relative speed. Let us note that the latter condition means that the implementation fits perfectly in its current speed. However, let us remark that this new notion corresponds neither to our first implementation relation (see Definition 7) nor to have a confidence level α equal to 1.

Definition 13 Let I and S be two TFSMs, H be a multiset of timed executions of I, and $0 \leq \alpha \leq 1$. We say that I $(\alpha, H)-$*stochastically and precisely conforms* to S, denoted by $I\ \mathtt{confp}^{(\alpha,H)}\ S$, if $I\ \mathtt{conf}_{s}^{(\alpha,H)}\ S$ and 1 is a relative speed of I with respect to S for H. □

The following result relates some of the notions presented in this section.

Lemma 4 Let I and S be two TFSMs. We have $I\ \mathtt{confp}^{(\alpha,H)}\ S$ iff $I\ \mathtt{conf}_{s}^{(\alpha,H)}\ S$ and neither I is generally faster than S for H nor I is generally slower than S for H. □

4.3. Time intervals implementation relations

In this case, time requirements are given by means of *intervals*. Next, we present our first timed implementation relation for this kind of systems.

Definition 14 Let I and S be two TFSMs. We say that I *conforms in time* to S, denoted by $I\ \mathtt{conf}_{int}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \mathtt{NTEvol}(I) \cap \mathtt{NTEvol}(S)$ we have that for all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e, \hat{t}) \in \mathtt{TEvol}(I) \implies (e, \hat{t}) \in \mathtt{TEvol}(S)$$

□

It is worth to point out that this relation suffers from the same practical problems as the relation introduced in Definition 7 for stochastic time. As we argued in the case of stochastic time, this notion of conformance, when used in a black-box testing framework, prevents us to ascertain if the intervals corresponding to the model and to the implementation are equal. In order to avoid this problem, we will apply a method similar to the one we have used in the previous section, based on a finite set of observations. However, we do not need to apply a contrast hypothesis: We will simply check that the observed time values belong to the specified time interval. Having this idea in mind, we will define three conformance relations where we check that the observed time values fulfill, in each case, the appropriate constraints.

Definition 15 Let I and S be two TFSMs, H be a multiset of timed executions of I, and $\Phi = \{e \mid \exists\ t : (e,t) \in H\} \cap \mathtt{NTEvol}(S)$. For all non-timed evolution $e \in \Phi$ we define the *sample interval* of e in H as $\widehat{S}_{(H,e)} = [\mathtt{min}(\mathtt{Sampling}_{(H,\Phi)}(e)), \mathtt{max}(\mathtt{Sampling}_{(H,\Phi)}(e))]$. We define the following implementation relations:

- I $H-$*timely conforms* to S, denoted by $I\ \mathtt{conf}^H_{int}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e, \hat{t}) \in \mathtt{TEvol}(S) \Longrightarrow \widehat{S}_{(H,e)} \subseteq \hat{t}$$

- I $H-$*fast timely conforms* to S, denoted by $I\ \mathtt{conf}^H_{intf}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e, \hat{t}) \in \mathtt{TEvol}(S) \Longrightarrow \widehat{S}_{(H,e)} \ll \hat{t}$$

- I $H-$*preferable timely conforms* to S, denoted by $I\ \mathtt{conf}^H_{intp}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ we have that for all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e, \hat{t}) \in \mathtt{TEvol}(S) \Longrightarrow \widehat{S}_{(H,e)} \preceq \hat{t}$$

□

Intuitively, the new relations establish that the implementation must conform to the specification in the usual way (that is, $I\ \mathtt{conf}_{nt}\ S$). In addition, the observed execution time values corresponding to an evolution must belong to the time interval indicated by the specification for that evolution (*timely conforms*), or be less than or equal to the lower/upper bound (*fast timely conforms*/*preferable timely conforms*) respectively. Let us remind that the relations between intervals $\subseteq$, $\ll$, and $\preceq$ were introduced in Definition 1.

5. Definition and application of tests

A test represents a sequence of inputs applied to the implementation. After applying each input, we check whether the received output is the expected one or not. In the latter case, a fail signal is produced. In the former case, either a pass signal is emitted (indicating successful termination) or the testing process continues by applying another input. If we are testing an implementation with input and output sets I and O, respectively, tests are deterministic acyclic I/O labelled transition systems (i.e. trees) with a strict alternation between an input action and the set of output actions. After an output action we may find either a leaf (indicating either failure or successful termination) or another input action. Leaves can be labelled either by *pass* or by *fail*. In the first case we add a *time stamp*. Depending on the kind of time requirements that are used to define the considered system, the time stamp will be a time value, a random variable, or a time interval. The idea is that we will record the time that the implementation takes to arrive to that point and compare it with the time stamp.

Definition 16 A *test* is a tuple $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C_T)$ where S is the set of states, I and O are disjoint sets of input and output actions, respectively, $Tr \subseteq S \times (I \cup O) \times S$ is the transition relation, $s_0 \in S$ is the initial state, and the sets $S_I, S_O, S_F, S_P \subseteq S$ are a partition of S. The transition relation and the sets of states fulfill the following conditions:

- S_I is the set of *input* states. We have that $s_0 \in S_I$. For all input state $s \in S_I$ there exists a unique outgoing transition $(s, a, s') \in Tr$. For this transition we have that $a \in I$ and $s' \in S_O$.

- S_O is the set of *output* states. For all output state $s \in S_O$ we have that for all $o \in O$ there exists a unique state s' such that $(s, o, s') \in Tr$. In this case, $s' \notin S_O$. Moreover, there do not exist $i \in I$ and $s' \in S$ such that $(s, i, s') \in Tr$.

- S_F and S_P are the sets of *fail* and *pass* states, respectively. We say that these states are *terminal*. That is, for all state $s \in S_F \cup S_P$ we have that there do not exist $a \in I \cup O$ and $s' \in S$ such that $(s, a, s') \in Tr$.

Finally, C_T is a function associating time stamps with passing states. Depending on the kind of time associated to the system that we are testing, the range of the function will correspond to the appropriate set. As expected, we will consider functions $C_T : S_P \longrightarrow \mathbb{R}_+$ for fix time values, $C_T : S_P \longrightarrow \mathcal{V}$ for time requirements expressed by random variables, and $C_T : S_P \longrightarrow \mathcal{I}_{\mathbb{R}_+}$ for time intervals.

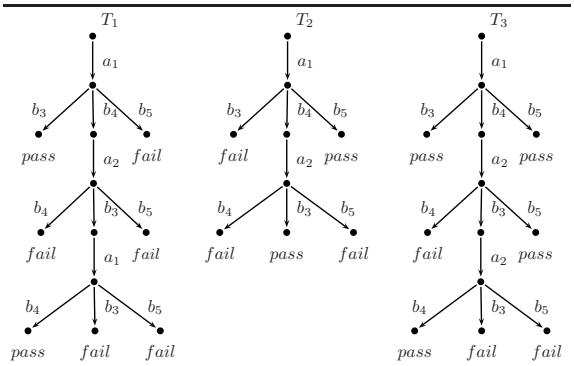


Figure 5. Examples of Test Cases: $I = \{a_1, a_2\}$ and $O = \{b_3, b_4, b_5\}$.

Let $e = i_1/o_1, \ldots, i_r/o_r$ and $s^T \in S_F \cup S_P$. We write $T \stackrel{e}{\Longrightarrow} s^T$ if there exist states $s_{12}, s_{21}, s_{22}, \ldots s_{r1}, s_{r2} \in S$ such that $\{(s_0, i_1, s_{12}), (s_{r2}, o_r, s^T)\} \subseteq Tr$, for all $2 \leq j \leq r$ we have $(s_{j1}, i_j, s_{j2}) \in Tr$, and for all $1 \leq j \leq r-1$ we have $(s_{j2}, o_j, s_{(j+1)1}) \in Tr$.

We say that a test case T is *valid* if the graph induced by T is a tree with root at the initial state s_0. We say that a set of tests $\mathcal{T}_{st} = \{T_1, \ldots, T_n\}$ is a *test suite*. □

In Figure 5 we present some examples of tests (in order to improve legibility, time stamps are omitted). From now on we will assume that when we talk about tests we refer only to valid tests. Next we define the application of a test to an implementation. We will say that the test suite $\mathcal{T}_{st}$ is *passed* if for all test the terminal states reached by the composition of implementation and test belong to the set of *passing* states.

Definition 17 Let I be a TFSM and T be a valid test. We denote the application of the test T to the implementation I by $I \parallel T$.

Let s^T be a state of T. We write $I \parallel T \stackrel{e}{\Longrightarrow} s^T$ if $T \stackrel{e}{\Longrightarrow} s^T$ and $e \in \mathtt{NTEvol}(I)$. Let $\mathcal{T}_{st}$ be a test suite. We say that I *passes* $\mathcal{T}_{st}$, denoted by $\mathtt{pass}(I, \mathcal{T}_{st})$, if for all test $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C_T) \in \mathcal{T}_{st}$ and $e \in \mathtt{NTEvol}(I)$ there do not exist $s^T \in S$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$ and $s^T \in S_F$. □

Let us remark that since we are assuming that implementations are input-enabled, the testing process will conclude only when the test reaches either a fail or a success state. In addition to this notion of passing tests, we will have different time conditions. Given the fact that this aspect depends on the three time domains that we are considering all along the paper, we will present them separately.

5.1. Fix time values

Regarding time expressed by means of fix values, we will consider two notions of passing tests. These two notions correspond to the implementation relations that we introduced in Section 4.1.

Definition 18 Let I be a TFSM, T be a test, and s^T be a state of T. We write $I \parallel T \stackrel{e}{\Longrightarrow}_t s^T$ if $T \stackrel{e}{\Longrightarrow} s^T$ and $(e, t) \in \mathtt{TEvol}(I)$.

Let $\sigma = (e, t) \in \mathtt{TEvol}(I)$ and $\mathcal{T}_{st}$ be a test suite. We define the set $\mathtt{Test}(\sigma, \mathcal{T}_{st}) = \{(T, s^T) \mid T \in \mathcal{T}_{st} \wedge I \parallel T \stackrel{e}{\Longrightarrow}_t s^T\}$.

We say that I *passes* the test suite $\mathcal{T}_{st}$ *for all time* if $\mathtt{pass}(I, \mathcal{T}_{st})$ and for all $\sigma = (e, t) \in \mathtt{TEvol}(I)$ and all $(T, s^T) \in \mathtt{Test}(\sigma, \mathcal{T}_{st})$ we have that $t = C(s^T)$ holds.

We say that I *passes* the test suite $\mathcal{T}_{st}$ *in the best time* if $\mathtt{pass}(I, \mathcal{T}_{st})$ and for all $\sigma = (e, t) \in \mathtt{TEvol}(I)$ and all $(T, s^T) \in \mathtt{Test}(\sigma, \mathcal{T}_{st})$ we have that $t \leq C(s^T)$ holds. □

5.2. Stochastic time

In this case we apply the time conditions to the set of *observed timed executions*, not to timed evolutions of the implementations, due to the fact that timed evolutions, in this domain, do not have a single time value that we can directly compare with the time stamp attached to the pass state. In fact, we need a set of test executions associated to each evolution in order to evaluate if they match the probability distribution function associated to the random variable indicated by the corresponding state of the test. In order to increase the degree of reliability, we will not take the classical approach where passing a test suite is defined according only to the results for each

test. In our approach, we will put together all the observations, for each test, so that we have more instances for each evolution. In particular, some observations will be used several times. In other words, an observation from a given test may be used to check the validity of another test sharing the same observed sequence. We will later show that this notion of passing tests is related to the implementation relation introduced in Definition 9.

Definition 19 Let I be a TFSM, T be a test, and s^T be a state of T. We write $I \| T \stackrel{e}{\Longrightarrow}_t s^T$ if $T \stackrel{e}{\Longrightarrow} s^T$ and (e,t) is an observed timed execution of I. In this case we say that (e,t) is a *test execution* of I and T.

Let I be a TFSM and $\mathcal{T}_{st} = \{T_1, \ldots, T_n\}$ be a test suite. Let $H_1, \ldots, H_n$ be test execution samples of I and $T_1, \ldots, T_n$, respectively. Let $H = \bigcup_{i=1}^{n} H_i$, $\Phi = \{e \mid \exists\, t : (e,t) \in H\}$, and let us consider $0 \leq \alpha \leq 1$. We say that I (α, H)-*passes* the test suite $\mathcal{T}_{st}$ if $\mathtt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, we have that $\gamma(C_T(s^T), \mathtt{Sampling}_{(H,\Phi)}(e)) > \alpha$. □

Let us note that an observed timed execution does not return the random variable associated with performing the evolution (that is, the addition of all the random variables corresponding to each transition of the implementation) but the time that it took to perform the evolution. Let us also note that in a fix time values framework, these two notions (addition of time values corresponding to the transitions of the implementation and observed time) do in fact coincide. Intuitively, an implementation passes a test if there does not exist an evolution leading to a fail state. Once we know that the functional behavior of the implementation is correct with respect to the test, we need to check time conditions. The set H corresponds to the observations of the (several) applications to I of the tests belonging to the test suite $\mathcal{T}_{st}$. In other words, observed timed executions will be used to conform the corresponding execution samples. Let us intuitively explain the process. We will apply each test belonging to the test suite to the implementation several times. If we find an unexpected output then we stop the testing process and conclude that the implementation is faulty without further checking its temporal behavior. If we do not find such an error, for each test we collect several observed timed executions corresponding to each time that the application of the test reached a pass state. Thus, we obtain for each test T_i a multiset $\{\!\{(e_1^i, t_1^i), (e_2^i, t_2^i), \ldots, (e_m^i, t_m^i)\}\!\}$. These multisets, more exactly the time values corresponding to each different evolution, will be used to make the hypothesis contrast. Thus, we have to decide whether, for each evolution e, the observed time values (that is, $\mathtt{Sampling}_{(H,\Phi)}(e)$) *match* the definition of the random variables appearing in the successful state of the tests corresponding to the execution of that evolution (that is, $C_T(s^T)$). As we previously commented, we assume a function γ, formally defined in the appendix, that can perform this hypothesis contrast.

Let us remark that the *quality* of the testing process depends not only on the considered test suite but also on the size of the execution samples. As we previously pointed out, the bigger these samples are, indicating that we were applying the tests more times without finding a functional error, the bigger will be the credibility of the testing process assessment. This is similar to the situation in testing non-deterministic systems where a fairness assumption cannot be made: We have to apply the same test several times (the more, the better) to increase the confidence that all the non-deterministic choices are exercised.

5.3. Time intervals

Finally, we consider the case when time is given by means of intervals. Following the idea presented in Section 4.3, we will have to check that the time values observed when executing tests (belonging to *sampling* sets) do fulfill the adequate conditions. We apply the notion of passing a test in the same way that we have presented for stochastic time. In fact, the notion of test execution introduced in Definition 19 is still valid and we do not repeat it. As we did in the stochastic setting, the set H corresponds again to the observations of the (several) applications of tests from the test suite $\mathcal{T}_{st}$ to I. We have defined different ways to establish if an implementation passes a

test suite, depending on the restrictions that the observed time values hold.

Definition 20 Let I be a TFSM and $\mathcal{T}_{st} = \{T_1, \ldots, T_n\}$ be a test suite. Let $H_1, \ldots, H_n$ be test execution samples of I and $T_1, \ldots, T_n$, respectively. Let $H = \bigcup_{i=1}^{n} H_i$ and $\Phi = \{e \mid \exists\, t : (e,t) \in H\}$. We say that

- I *H-passes in time* the test suite $\mathcal{T}_{st}$ if $\mathtt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, we have that $\widehat{S}_{(H,e)} \subseteq C_T(s^T)$.
- I *H-passes fast* the test suite $\mathcal{T}_{st}$ if $\mathtt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, we have that $\widehat{S}_{(H,e)} \ll C_T(s^T)$.
- I *H-passes preferable* the test suite $\mathcal{T}_{st}$ if $\mathtt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, we have that $\widehat{S}_{(H,e)} \preceq C_T(s^T)$.

□

6. Derivation of test suites

In this section we present an algorithm to derive tests from TFSMs. These test suites are sound and complete with respect to the implementation relations introduced in Section 4.

6.1. Derivation algorithm

The basic idea underlying test derivation consists in traversing the specification in order to get all the possible evolutions in an appropriate way. First, we introduce some additional notation.

Definition 21 Let $M = (S, I, O, Tr, s_{in})$ be a TFSM. We define the function $\mathtt{out} : S \times I \longrightarrow \mathcal{P}(O)$ such that for all $s \in S$ and $i \in I$ it returns the set of outputs

$$\mathtt{out}(s,i) = \{o \mid \exists\, s', d :\ (s, s', i, o, d) \in Tr\}$$

We define the function $\mathtt{after} : S \times I \times O \times \mathcal{T}\mathtt{ime} \longrightarrow ((S \times \mathcal{T}\mathtt{ime}) \cup \{\mathtt{error}\})$ such that for all $s \in S, i \in I, o \in O$, and $d \in \mathcal{T}\mathtt{ime}$ we have

$$\mathtt{after}(s,i,o,d) = \begin{cases} (s', d + d') \text{ if } (s, s', i, o, d') \in Tr \\ \mathtt{error} \qquad \text{otherwise} \end{cases}$$

□

The function $\mathtt{out}(s,i)$ computes the set of output actions associated with those transitions that can be executed from s after receiving the input i. The function $\mathtt{after}(s,i,o,d)$ computes the *situation* that is reached from a state s after receiving the input i, producing the output o, when the duration of the previous testing process is d. By *situation* we mean a pair containing the reached state and the cumulated duration since the system started its performance. Let us also remark that due to the assumption that TFSMs are observable we have that $\mathtt{after}(s,i,o,d)$ is uniquely determined. Besides, let us note that the addition of time values given by the expression $d + d'$ will denote, depending on the machine, the addition of two real numbers, two random variables, or two intervals. Finally, we will apply this function only when the side condition holds, that is, we will never receive error as result of applying after.

The algorithm to derive tests from a specification is given in Figure 6. By considering the possible available choices we get a test suite extracted from M. We denote this test suite by $\mathtt{tests}(M)$. Next we explain how our algorithm works. A set of *pending situations* S_{aux} keeps those tuples denoting the possible states and duration values that could appear in a state of the test whose outgoing transitions have not been completed yet. More precisely, a tuple $(s^M, d, s^T) \in S_{aux}$ indicates that we did not complete the state s^T of the test, the current state in the traversal of the specification is s^M, and the accounting for the elapsed duration in the specification from the initial state is given by d.

Our algorithm can be applied in the different domains we consider along the work. For each of them, we use an appropriate time function associated to the kind of machine to which we apply the algorithm. That is, the accumulated duration (i.e. the value d) will be a time value, a random variable, or a time interval.

Following with the explanation of the algorithm, the set S_{aux} initially contains a tuple with the initial states (of both TFSM and test) and the initial situation of the process (that is, duration

Input: A `TFSM` $M = (S, I, O, Tran, s_{in})$.
Output: A test case $T = (S', I, O, Tran', s_0, S_I, S_O, S_F, S_P, C_T)$.

Initialization: $S' := \{s_0\}$; $Tran', S_I, S_O, S_F, S_P := \emptyset$; $S_{aux} := \{(s_{in}, \texttt{zero}, s_0)\}$.
Inductive Cases: Choose one of the following two options until $S_{aux} = \emptyset$.

1. If $(s^M, d, s^T) \in S_{aux}$ then perform the following steps:
   {s^T will be a pass state; time values computed while applying the}
   {test and reaching this state will be *compared* with d}
   (a) $S_{aux} := S_{aux} - \{(s^M, d, s^T)\}$; $S_P := S_P \cup \{s^T\}$; $C_T(s^T) := d$.
2. If $S_{aux} = \{(s^M, d, s^T)\}$ and $\exists\, i \in I : \texttt{out}(s^M, i) \neq \emptyset$ then perform the following steps:
   (a) $S_{aux} := \emptyset$; Choose i such that $\texttt{out}(s^M, i) \neq \emptyset$.
   (b) Consider a fresh state $s' \notin S'$ and let $S' := S' \cup \{s'\}$.
   (c) $S_I := S_I \cup \{s^T\}$; $S_O := S_O \cup \{s'\}$; $Tran' := Tran' \cup \{(s^T, i, s')\}$.
   {Add an input transition labelled by i and consider all outputs}
   (d) For all $o \notin \texttt{out}(s^M, i)$ do {These outputs lead to a fail state}
   - Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
   - $S_F := S_F \cup \{s''\}$; $Tran' := Tran' \cup \{(s', o, s'')\}$.

   (e) For all $o \in \texttt{out}(s^M, i)$ do
   {These outputs are *expected*. At most one of them will lead to an input state}
   {where the test continues; the rest will lead to pass states}
   - Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
   - $Tran' := Tran' \cup \{(s', o, s'')\}$.
   - $(s_1^M, d') := \texttt{after}(s^M, i, o, d)$; $S_{aux} := S_{aux} \cup \{(s_1^M, d', s'')\}$.

Figure 6. Derivation of tests from an observable specification.

`zero`). If we are extracting tests from a `TFSM` where time is given by fix time values, `zero` denotes that the elapsed time is $0 \in \mathbb{R}_+$. If we consider time given by using random variables, `zero` denotes a random variable ξ following a Dirac distribution in 0, that is, $F_\xi(x) = 1$ for all $x \in \mathbb{R}_+$. Finally, in the case of time intervals we mean $[0, 0]$. For each tuple belonging to S_{aux} we may choose one possibility. It is important to remark that the second step can be applied only when the set S_{aux} becomes singleton. So, our derived tests correspond to valid tests as introduced in Definition 16. The first possibility simply indicates that the state of the test becomes a passing state. The second possibility takes an input and generates a transition in the test labelled by this input. Then, the whole sets of outputs is considered. If the output is not expected by the `TFSM` (step 2.(e) of the algorithm) then a transition leading to a failing state is created. This could be simulated by a single branch in the test, labelled by `else`, leading to a failing state (in the algorithm we suppose that *all* the possible outputs appear in the test). For the expected outputs (step 2.(f) of the algorithm) we create a transition with the corresponding output and add the appropriate tuple to the set S_{aux}.

Let us note that finite tests are constructed simply by considering a step where the second inductive case is not applied.

Let us comment on the *finiteness* of our algorithm. If we do not impose any restriction on the implementation (e.g. a bound on the number of states) we cannot determine some important information such as the maximal length of the traces that the implementation can perform. In other words, as commented more thoroughly in the forthcoming Section 8 devoted to related work, we would need a *fault coverage criterion* to generate a finite test suite. Actually, this is also the case in work related to ours such as the `ioco` theory [29]. Obviously, one can impose restrictions such as "generate n tests" or "generate all the tests with m inputs" and *completeness* will be obtained up to that coverage criterion. Since we do not assume, by default, any criteria, all we can do is to say that this is the, in general, test suite that allows to prove completeness, that is, we obtain full fault coverage but taking into account that the derived test suite will be, in general, infinite. This can be seen in the forthcoming Theorems 1, 2, and 3 where we show that an implementation conforms to a formal model iff the implementation passes all the tests belonging to the test suite. Finally, let us remark that even if the derived test suite is infinite, it can still be taken as a first step to generate finite test suites (contained in it) with respect to a given fault coverage criterion. In this line, it is very easy to consider a fault coverage criterion based on checking the behavior of systems up to a given length. That is, in order to ensure that the implementation under test is correct in its *first* n steps (that is, for input/output sequences having length less than or equal to n), it is enough to consider the subset of the derived test suite including those tests having that length. However, the topic of adapting our framework to deal with this issue is outside the scope of this paper.

Regarding the computational complexity of the algorithm we have to distinguish two cases. If the specification has loops then the derived test suite is infinite. This is due again to the fact that we do not assume any hypothesis regarding the implementation where these tests will be applied. Thus, we cannot limit the size of the tests. If the specification does not have loops then the number of tests appearing in the test suite is, in the worst case, exponential with respect to the number of states of the specification. The idea is that the test suite contains all the possible paths/sequences that can be performed from the initial state of the specification. In order to find a fault in the implementation we have to detect that one of these feasible paths is not performed in the expected way, that is, after applying one of the inputs conforming the sequence we receive an unexpected output. Let us remark that these numbers are usual in test derivation algorithms based on `ioco`.

6.2. Soundness and completeness

Next, we present the results that relate implementation relations and application of test suites derived from a model described as a `TFSM`. The non-timed aspects of our algorithm are based on the algorithm developed for the `ioco` relation. So, in spite of the differences, the non-timed part of the proof of our result is a simple adaptation of that in [29]. Regarding the part of the proof related to time issues, the result holds because the temporal conditions required to conform to the model and to pass the test suite are in fact the same. We will give the detailed proof for the implementation relations for machines presenting time requirements expressed by means of fix time values. Next, we will sketch how the result can be adapted for the other two notions of time considered in this paper.

Theorem 1 Let S and I be two `TFSM`s. We have that:

- I $\texttt{conf}_a$ S iff I passes $\texttt{tests}(S)$ for all time.
- I $\texttt{conf}_b$ S iff I passes $\texttt{tests}(S)$ in the best time.

Proof: We will only prove the first result since the technique is similar for the second one. First, let us show that I passes $\texttt{tests}(S)$ for all time implies I $\texttt{conf}_a$ S. We will use the contrapositive, that is, we will suppose that I $\texttt{conf}_a$ S does not hold and we will prove that I does not pass

$\texttt{tests}(S)$ for all time. If $I\ \texttt{conf}_a\ S$ does not hold then we have two possibilities:

- Either $I\ \texttt{conf}_{nt}\ S$ does not hold, or
- there exists a temporal evolution $(e,t) \in \texttt{TEvol}(I)$ such that $e \in \texttt{NTEvol}(S)$ and $(e,t) \notin \texttt{TEvol}(S)$.

Let us consider the first case, that is, we suppose that $I\ \texttt{conf}_{nt}\ S$ does not hold. Then, there exist two non-timed evolutions $e = (i_1/o_1, \ldots, i_r/o_r)$ and $e' = (i_1/o_1, \ldots, i_r/o'_r)$, with $r \geq 1$, such that $e \in \texttt{NTEvol}(S)$, $e' \in \texttt{NTEvol}(I)$, and $e' \notin \texttt{NTEvol}(S)$. We have to show that there exists a test $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C_T) \in \texttt{tests}(S)$ such that $T \stackrel{e}{\Longrightarrow} s^T$, with $s^T \in S_P$, and $T \stackrel{e'}{\Longrightarrow} u^T$, with $u^T \in S_F$. In this situation, we would conclude I does not pass $\texttt{tests}(S)$. We can construct this test by applying the algorithm given in Figure 6 and by resolving the non-deterministic choices in the following way:

```
for 1 ≤ j ≤ r do
  • apply 2nd inductive case for input i_j
  • apply 1st inductive case for all elements
    (s^M, d, s^T) ∈ S_aux obtained by processing
    an output different from o_j
endfor
apply 1st inductive case for last element
  (s^M, d, s^T) ∈ S_aux
```

The previous algorithm generates a test T such that $T \stackrel{e'}{\Longrightarrow} u^T$, with $u^T \in S_F$. This is so because the last application of the second inductive case for the output o'_r must be necessarily associated to the step 2.(e) since $e' \notin \texttt{NTEvol}(S)$. In addition, we have that if $e' \in \texttt{NTEvol}(I)$ then $I \parallel T \stackrel{e'}{\Longrightarrow} u^T$. Given the fact that $T \in \texttt{tests}(S)$ we deduce that $\texttt{pass}(I, \texttt{tests}(S))$ does not hold. Thus, we conclude I does not pass $\texttt{tests}(S)$ for all time.

Let us suppose now that $I\,\texttt{conf}_a\,S$ does not hold because there exists a temporal evolution $(e,t) \in \texttt{TEvol}(I)$ such that $e \in \texttt{NTEvol}(S)$ but $(e,t) \notin \texttt{TEvol}(S)$. Let us consider the same test T that we defined before by taking into consideration the evolution e. Since $e \in \texttt{NTEvol}(S)$ we have that $T \stackrel{e}{\Longrightarrow} s^T$, with $s^T \in S_P$. Besides, since $(e,t) \in \texttt{TEvol}(I)$, we also have $I \parallel T \stackrel{e}{\Longrightarrow}_t s^T$. If $(e,t) \notin \texttt{TEvol}(S)$ then $t \neq C_T(s^T)$. We conclude I does not pass $\texttt{tests}(S)$ for all time.

Next we prove that $I\ \texttt{conf}_a\ S$ implies I passes $\texttt{tests}(S)$ for all time. We will use again the contrapositive, that is, we will assume that I does not pass $\texttt{tests}(S)$ for all time and we will conclude that $I\ \texttt{conf}_a\ S$ does not hold. If I does not pass $\texttt{tests}(S)$ for all time then we have two possibilities:

- Either $\texttt{pass}(I, \texttt{tests}(S))$ does not hold, or
- there exists $(e,t) \in \texttt{TEvol}(I)$ and $(T, s^T) \in \texttt{Test}(e, \texttt{tests}(S))$ such that $I \parallel T \stackrel{e}{\Longrightarrow}_t s^T$, with $s^T \in S_P$, and $t \neq C_T(s^T)$.

First, let us assume that I does not pass $\texttt{tests}(S)$ for all time because $\texttt{pass}(I, \texttt{tests}(S))$ does not hold. This means that there exists a test $T \in \texttt{tests}(S)$, a non-timed evolution $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r)$, and $s^T \in S_F$ fulfilling $I \parallel T \stackrel{e}{\Longrightarrow} s^T$. Then, we have $e \in \texttt{NTEvol}(I)$ and $T \stackrel{e}{\Longrightarrow} s^T$. According to our derivation algorithm, a branch of a derived test leads to a fail states only if its associated output action is not expected in the specification. Thus, $e \notin \texttt{NTEvol}(S)$. Let us note that our algorithm allows to create a fail state only as the result of the application of the second inductive case. One of the premises of this inductive case is $\texttt{out}(S^M, i) \neq \emptyset$, that is, the specification is allowed to perform some output actions after the reception of the corresponding input. Thus, there exists an output action o'_r and an evolution $e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r)$ such that $e' \in \texttt{NTEvol}(S)$. Given the fact that $e \in \texttt{NTEvol}(I)$, $e \notin \texttt{NTEvol}(S)$, and $e' \in \texttt{NTEvol}(S)$, we have that $I\ \texttt{conf}_{nt}\ S$ does not hold. We conclude $I\ \texttt{conf}_a\ S$ does not hold.

Let us suppose now that I does not pass $\texttt{tests}(S)$ for all time because there exist $(e,t) \in \texttt{TEvol}(I)$ and $(T, s^T) \in \texttt{Test}(e, \texttt{tests}(S))$ such that $I \parallel T \stackrel{e}{\Longrightarrow}_t s^T$, with $s^T \in S_P$ and $t \neq C(s^T)$. Since $s^T \in S_P$ we deduce $e \in \texttt{NTEvol}(S)$. Besides, since $t \neq C(s^T)$, we deduce $(e,t) \notin \texttt{TEvol}(S)$. Finally, by taking into account that

$(e,t) \in \texttt{TEvol}(I)$, we conclude that $I\,\texttt{conf}_a\,S$ does not hold. □

Regarding systems with time requirements given by using random variables, we will show that the derived test suite is also *sound* and *complete*, up to a given confidence level α and for a sample H, with respect to the conformance relation $\texttt{conf}_s^{(\alpha,H)}$. Specifically, the next result states that for a given specification S, the test suite $\texttt{tests}(S)$ can be used to distinguish those (and only those) implementations that conform with respect to $\texttt{conf}_s^{(\alpha,H)}$. However, we cannot say that the test suite is complete since both the notion of passing tests and the considered implementation relation have a probabilistic component. So, we can talk of *completeness* up to a certain confidence level.

Theorem 2 Let I and S be two TFSMs. For all $0 \leq \alpha \leq 1$ and multiset of timed executions H we have $I\,\texttt{conf}_s^{(\alpha,H)}\,S$ iff I $(\alpha,H)-$*passes* $\texttt{tests}(S)$.

Proof Sketch: $I\,\texttt{conf}_s^{(\alpha,H)}\,S$ requires $I\,\texttt{conf}_{nt}\,S$ and that for all evolution belonging to both S and I the confidence of the random variable appearing in S on the samples observed in I is higher than α. The way our algorithm deals with non-stochastic information of the evolutions is independent of the specific notion of time that we are considering. Thus, this part of the proof is the same as the proof of Theorem 1. Regarding stochastic information, let us remark that the samples collected from the tests will be exactly the ones we apply to check whether the implementation relation holds. So, the conditions we require about the confidence of the random variables on the samples will be the same both to pass the test and to make the implementation relation to hold. □

Next, we present a result to establish the application of the test suite $\texttt{tests}(S)$ for determining whether an implementation I, for a sample H, conforms to the model S with respect to the relations $\texttt{conf}_{int}^H$, $\texttt{conf}_{intf}^H$, and $\texttt{conf}_{intp}^H$ given in Definition 15.

Theorem 3 Let I and S be two TFSMs. Given a multiset of timed executions H we have

- $I\,\texttt{conf}_{int}^H S$ iff I $H-$*passes in time* $\texttt{tests}(S)$.
- $I\,\texttt{conf}_{intf}^H\,S$ iff I $H-$*passes fast* $\texttt{tests}(S)$.
- $I\,\texttt{conf}_{intp}^H\,S$ iff I $H-$*passes preferable* $\texttt{tests}(S)$.

Proof Sketch: $I\,\texttt{conf}_{int}^H\,S$ requires $I\,\texttt{conf}_{nt}\,S$ and that for all evolution belonging to both S and I, the samples observed in the implementation belong to the time interval considered in S. In the same way that we have followed for stochastic time, the part of the proof corresponding to non-temporal information coincides with the proof of Theorem 1. Regarding temporal information, again, the samples collected from the tests are the samples we apply to check whether the implementation relation holds and the requirements established to pass the tests are the same as the ones demanded by the implementation relation. □

7. Considering non-deterministic behaviors

In this section we will extend the behavior of TFSMs to allow them to present full non-deterministic behaviors. That is, we allow a machine to have two different transitions as (s,s_1,i,o,d_1) and (s,s_2,i,o,d_2). This fact will increment the number of implementations relations that we introduced in Section 4.

The new relations are defined by taking into account the fact that a non-timed evolution can be performed in different time values depending on the trace we consider and the time function associated with the transitions that take part of it. It is interesting to mention that non-determinism does not give raise to new implementation relations in the case of systems having time specified by using random variables. This is due to the fact that random variables will be combined by means of an appropriate operator (e.g. the minimum if a race policy is used). In other words, if we have two transitions such as $s \xrightarrow{i/o}_{\xi_1} s_1$ and $s \xrightarrow{i/o}_{\xi_2} s_2$, this situation is equivalent to say that s can perform i/o by following a random variable distributed as $\texttt{min}(\xi_1,\xi_2)$. This is

not the case, for example, if we have fix values: If we have two transitions such as $s \xrightarrow{i/o}_{t_1} s_1$ and $s \xrightarrow{i/o}_{t_2} s_2$, we just can say that i/o is performed either in time t_1 or in time t_2, but we can not combine these two values. In conclusion, by allowing non-determinism in systems where time is expressed by means of random variables we do not get additional implementation relations. Thus, in this part of the work we will consider that time is given either by means of fix values or by time intervals.

7.1. Timed implementation relations

The $\texttt{conf}_a$ relation (conforms *always*) defined in Section 4.1 remains the same. That is, for all timed evolution (e,t) of the implementation we have that if e is a non-timed evolution of the model then (e,t) is also a timed evolution of the specification. The $\texttt{conf}_b$ relation (conforms in the *best* case) is similar to $\texttt{conf}_a$ but considering the fastest instance. In the $\texttt{conf}_w$ relation (conforms in the *worst* case) the implementation is forced, for each timed evolution fulfilling the previous conditions, to be faster than the slowest instance of the same evolution of the model. The $\texttt{conf}_{sw}$ relation requests that, for each of its evolutions, at least one instance of the implementation must be faster than the slowest instance, for the same evolution, of the model. The $\texttt{conf}_{sb}$ relation requests that, for each of its evolutions, at least one instance of the implementation is faster than the fastest instance of the model. The notion of *instance* can be formally defined as follows: Given a machine M and a non-timed evolution $e \in \texttt{NTEvol}(M)$, for all $t \in \mathcal{T}\texttt{ime}$ such that $(e,t) \in \texttt{TEvol}(M)$, we say that (e,t) is an instance of e.

Definition 22 Let S and I be two TFSMs. We define the following implementation relations:

- *(always)* $I\ \texttt{conf}_a\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time value $t \in \mathbb{R}_+$

$$(e,t) \in \texttt{TEvol}(I) \Longrightarrow (e,t) \in \texttt{TEvol}(S)$$

- *(best)* $I\ \texttt{conf}_b\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time value $t \in \mathbb{R}_+$

$$\begin{array}{c}(e,t) \in \texttt{TEvol}(I)\\ \Downarrow \\ \forall\, t' \in \mathbb{R}_+ : ((e,t') \in \texttt{TEvol}(S) \Longrightarrow t \leq t')\end{array}$$

- *(worst)* $I\ \texttt{conf}_w\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time value $t \in \mathbb{R}_+$

$$\begin{array}{c}(e,t) \in \texttt{TEvol}(I)\\ \Downarrow \\ \exists\, t' \in \mathbb{R}_+ : ((e,t') \in \texttt{TEvol}(S)\ \wedge\ t \leq t')\end{array}$$

- *(sometimes best)* $I\ \texttt{conf}_{sb}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that there exists a time value $t \in \mathbb{R}_+$ such that

$$\begin{array}{c}(e,t) \in \texttt{TEvol}(I)\\ \wedge \\ \forall\, t' \in \mathbb{R}_+ : ((e,t') \in \texttt{TEvol}(S) \Longrightarrow t \leq t')\end{array}$$

- *(sometimes worst)* $I\ \texttt{conf}_{sw}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that there exist time values $t, t' \in \mathbb{R}_+$ such that

$$\begin{array}{c}(e,t) \in \texttt{TEvol}(I)\\ \wedge \\ (e,t') \in \texttt{TEvol}(S)\ \wedge\ t \leq t'\end{array}$$

□

Theorem 4 The relations given in Definition 22 are related as follows:

$$\begin{array}{ccccc} & & I\ \texttt{conf}_b\ S & \Rightarrow & I\ \texttt{conf}_{sb}\ S \\ & & \Downarrow & & \Downarrow \\ I\ \texttt{conf}_a\ S & \Rightarrow & I\ \texttt{conf}_w\ S & \Rightarrow & I\ \texttt{conf}_{sw}\ S \end{array}$$

Proof: We only need to consider the evolutions of I belonging also to S (for the rest of evolutions, the premises of the corresponding conformance relation do not hold). First, let us note that the condition about non-timed conformance is the same in all the notions. So, we only need to take into account the conditions on time appearing in the second clause of the corresponding relations. If $I\ \texttt{conf}_a\ S$ then we have that each timed evolution in I fulfilling the conditions given

in the definition of $\mathtt{conf}_a$ does also appear in S. Thus, we have $I\ \mathtt{conf}_w\ S$. If $I\ \mathtt{conf}_b\ S$ then each timed evolution of I fulfilling the conditions given in the definition of $\mathtt{conf}_b$ is faster than the fastest instance of the same evolution for S. Therefore, it is also faster than the slowest one for S, and so $I\ \mathtt{conf}_w\ S$.

If $I\ \mathtt{conf}_w\ S$ then we know that each instance of a temporal evolution of I needs a time less than or equal to the one corresponding to the slowest instance, for the same evolution, of the specification S. In particular, there exists an instance fulfilling the condition imposed by $\mathtt{conf}_{sw}$. So, we conclude $I\ \mathtt{conf}_{sw}\ S$. The same reasoning can be also used to prove that $I\ \mathtt{conf}_b\ S$ implies $I\ \mathtt{conf}_{sb}\ S$.

Finally, we have to study the relation between $\mathtt{conf}_{sb}$ and $\mathtt{conf}_{sw}$. If $I\ \mathtt{conf}_{sb}\ S$ then we have that for all evolution of I there exists an instance being faster than the fastest instance of the same evolution in S. In particular, this instance is also faster than the slowest instance of S, so that we conclude $I\ \mathtt{conf}_{sw}\ S$. □

It is interesting to note that if specifications are restricted to take always the same time for each given evolution (independently from the possible derivation taken for such evolution) then the relations $\mathtt{conf}_b$, $\mathtt{conf}_w$, $\mathtt{conf}_{sw}$, and $\mathtt{conf}_{sb}$ would coincide, but they would be still different from the $\mathtt{conf}_a$ relation. This is so because $\mathtt{conf}_a$ requires that time values in the implementation coincide with those in the specification, while other relations require that time values in the implementation are *less than or equal to* those of the specification.

Lemma 5 Let I and S be TFSMs. We have the following results:

(1) If for all non-temporal evolution $(i_1/o_1, \ldots, i_r/o_r) \in \mathtt{NTEvol}(S)$ there do not exist two different time values $t, t' \in \mathbb{R}_+$ such that $((i_1/o_1, \ldots, i_r/o_r), t)$ and $((i_1/o_1, \ldots, i_r/o_r), t')$ belong to $\mathtt{TEvol}(S)$, then $I\ \mathtt{conf}_w\ S$ iff $I\ \mathtt{conf}_b\ S$ and $I\ \mathtt{conf}_{sw}\ S$ iff $I\ \mathtt{conf}_{sb}\ S$.

(2) If for all non-temporal evolution $(i_1/o_1, \ldots, i_r/o_r) \in \mathtt{NTEvol}(I)$ there do not exist two different time values $t, t' \in \mathbb{R}_+$ such that $((i_1/o_1, \ldots, i_r/o_r), t)$ and $((i_1/o_1, \ldots, i_r/o_r), t')$ belong to $\mathtt{TEvol}(S)$, then $I\ \mathtt{conf}_w\ S$ iff $I\ \mathtt{conf}_{sw}\ S$ and $I\ \mathtt{conf}_b\ S$ iff $I\ \mathtt{conf}_{sb}\ S$.

(3) If the conditions of the results (1) and (2) hold then the relations $\mathtt{conf}_w$, $\mathtt{conf}_b$, $\mathtt{conf}_{sw}$, and $\mathtt{conf}_{sb}$ coincide.

Proof: If the condition in (1) holds then the best instance of each evolution of S is actually the worst instance of that evolution. Similarly, if the condition in (2) holds then the best instance of each evolution of I is the worst one as well. The last result is obtained from results (1) and (2) by applying transitivity between relations. □

7.2. Time intervals implementation relations

In this section we present our timed implementation relations for systems where time requirements are expressed by means of time intervals. We begin by giving several variations of the relation presented in Definition 14, $\mathtt{conf}_{int}$. In the $\mathtt{conf}_w^{int}$ relation (conforms in the *worst* case) we force the implementation to be faster than the model for some of its timed evolutions. In other words, the lower bound of the time interval of the model is greater than the upper bound of the one corresponding to the implementation. The $\mathtt{conf}_b^{int}$ relation (conforms in the *best* case) is similar, but considering all timed evolutions of the model.

We also consider an intermediate case: *preferable* conform relations. In this kind of relations we have that the timed evolutions of the implementation are faster than the slowest timed evolution of the model, but probably slower than the fastest one. We distinguish among the different levels by taking into account if this condition holds for all the timed evolutions ($\mathtt{conf}_p^{int}$, $\mathtt{conf}_{mp}^{int}$, and $\mathtt{conf}_{lp}^{int}$) or only for some of them ($\mathtt{conf}_{smp}^{int}$ and $\mathtt{conf}_{slp}^{int}$).

Definition 23 Let S and I be two TFSMs. We define the following implementation relations:

- *(best)* $I\ \mathtt{conf}_b^{int}\ S$ iff $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \mathtt{NTEvol}(I) \cap \mathtt{NTEvol}(S)$ we have that for

all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\Downarrow$$
$$\forall\, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+} : ((e,\hat{t'}) \in \texttt{TEvol}(S) \Longrightarrow \hat{t} \ll \hat{t'})$$

- *(worst)* $I\ \texttt{conf}^{int}_{w}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\Downarrow$$
$$\exists\, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+} : ((e,\hat{t'}) \in \texttt{TEvol}(S)\ \wedge\ \hat{t} \ll \hat{t'})$$

- *(sometimes best)* $I\ \texttt{conf}^{int}_{sb}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that there exists a time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$ such that

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\wedge$$
$$\forall\, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+} : ((e,\hat{t'}) \in \texttt{TEvol}(S) \Longrightarrow \hat{t} \ll \hat{t'})$$

- *(sometimes worst)* $I\ \texttt{conf}^{int}_{sw}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that there exist time intervals $\hat{t}, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+}$ such that

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\wedge$$
$$(e,\hat{t'}) \in \texttt{TEvol}(S)\ \wedge\ \hat{t} \ll \hat{t'}$$

- *(preferable)* $I\ \texttt{conf}^{int}_{p}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\Downarrow$$
$$\forall\, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+} : ((e,\hat{t'}) \in \texttt{TEvol}(S) \Longrightarrow \hat{t} \subseteq \hat{t'})$$

- *(more preferable)* $I\ \texttt{conf}^{int}_{mp}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\Downarrow$$
$$\forall\, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+} : (e,\hat{t'}) \in \texttt{TEvol}(S) \Longrightarrow \hat{t} \preceq \hat{t'})$$

- *(less preferable)* $I\ \texttt{conf}^{int}_{lp}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that for all time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\Downarrow$$
$$\exists\, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+} : ((e,\hat{t'}) \in \texttt{TEvol}(S)\ \wedge\ \hat{t} \preceq \hat{t'})$$

- *(sometimes more preferable)* $I\ \texttt{conf}^{int}_{smp}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that there exists a time interval $\hat{t} \in \mathcal{I}_{\mathbb{R}_+}$ such that

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\wedge$$
$$\forall\, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+} : ((e,\hat{t'}) \in \texttt{TEvol}(S) \Longrightarrow t \preceq t')$$

- *(sometimes less preferable)* $I\ \texttt{conf}^{int}_{slp}\ S$ iff $I\ \texttt{conf}_{nt}\ S$ and for all $e \in \texttt{NTEvol}(I) \cap \texttt{NTEvol}(S)$ we have that there exist time intervals $\hat{t}, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+}$ such that

$$(e,\hat{t}) \in \texttt{TEvol}(I)$$
$$\wedge$$
$$(e,\hat{t'}) \in \texttt{TEvol}(S)\ \wedge\ \hat{t} \preceq \hat{t'})$$

□

Let us remind again that the relations on time intervals $\subseteq$, $\ll$, and $\preceq$ were introduced in Definition 1.

The previously defined implementation relations have the same practical drawbacks as we commented before: In a black-box framework we cannot *see* the time intervals appearing in the implementation. However, these relations are interesting and useful, in fact, if we would work in a white-box framework. In such framework, we can establish different levels of conformance, taking into account that we have information about the time intervals that appear in the implementation. So, we can evaluate them with respect to the ones that are established in the model. Because of its interest in a white-box setting, and having in mind that they are the starting point for the forthcoming relations given in Definition 24, we have presented here these relations and we will study some properties.

Theorem 5 The relations given in Definitions 14 and 23 are related as follows:

$$\begin{array}{ccc} I\ \mathtt{conf}_{b}^{int}\ S & \Rightarrow & I\ \mathtt{conf}_{sb}^{int}\ S \\ \Downarrow & & \Downarrow \\ I\ \mathtt{conf}_{w}^{int}\ S & \Rightarrow & I\ \mathtt{conf}_{sw}^{int}\ S \end{array}$$

$$\begin{array}{ccc} I\ \mathtt{conf}_{int}\ S & \Rightarrow & I\ \mathtt{conf}_{p}^{int}\ S \\ \Downarrow & & \\ I\ \mathtt{conf}_{lp}^{int}\ S & \Rightarrow & I\ \mathtt{conf}_{slp}^{int}\ S \\ \Uparrow & & \Uparrow \\ I\ \mathtt{conf}_{mp}^{int}\ S & \Rightarrow & I\ \mathtt{conf}_{smp}^{int}\ S \end{array}$$

Proof: We consider the evolutions of I belonging also to S. As in the case of conformance relations for fix time values, the condition about the non-timed conformance is the same in all the definitions. So we will only analyze temporal conditions. If $I\ \mathtt{conf}_{b}^{int}\ S$ then each timed evolution of I fulfilling the conditions given in the definition of $\mathtt{conf}_{b}^{int}$ is faster than the fastest instance of the same evolution for S. Therefore, it is also faster than the slowest one for S. Thus, we conclude, $I\ \mathtt{conf}_{w}^{int}\ S$.

If $I\ \mathtt{conf}_{w}^{int}\ S$ then we know that each instance of a temporal evolution of I needs a time less than or equal to the one corresponding to the slowest instance, for the same evolution, of S. In particular, there exists an instance fulfilling the condition imposed by $\mathtt{conf}_{sw}^{int}$. So, we conclude $I\ \mathtt{conf}_{sw}^{int}\ S$. The same reasoning can be also used to prove that $I\ \mathtt{conf}_{b}^{int}\ S$ implies $I\ \mathtt{conf}_{sb}^{int}\ S$.

Regarding the relation between $\mathtt{conf}_{sb}^{int}$ and $\mathtt{conf}_{sw}^{int}$, if $I\ \mathtt{conf}_{sb}^{int}\ S$ then we have that for all evolution of I there exists an instance being faster than the fastest instance of the same evolution in S. In particular, this instance is also faster than the slowest instance of S, so that we conclude $I\ \mathtt{conf}_{sw}^{int}\ S$.

The reasoning for the relations $\mathtt{conf}_{lp}^{int}$, $\mathtt{conf}_{mp}^{int}$, $\mathtt{conf}_{slp}^{int}$, and $\mathtt{conf}_{smp}^{int}$ is similar to the one used for the relations $\mathtt{conf}_{w}^{int}$, $\mathtt{conf}_{b}^{int}$, $\mathtt{conf}_{sw}^{int}$, and $\mathtt{conf}_{sb}^{int}$.

Now, we have to study *always* and *preferable* relations. First, we have that if $I\ \mathtt{conf}_{int}\ S$ then each timed evolution in I fulfilling the conditions given in the definition of $\mathtt{conf}_{int}$ does also appear in S, so we have $I\ \mathtt{conf}_{p}^{int}\ S$. Using the same reasoning we have that if $I\ \mathtt{conf}_{int}\ S$ holds then $\mathtt{conf}_{lp}^{int}$ holds too. □

In the next result we show how the relations collapse if we restrict ourselves to observable machines.

Lemma 6 Let I and S be two TFSMs. We have the following results:

(1) If for all non-temporal evolution $(i_1/o_1, \ldots, i_r/o_r) \in \mathtt{NTEvol}(S)$ there do not exist two different intervals $\hat{t}, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+}$ such that $((i_1/o_1, \ldots, i_r/o_r), \hat{t})$ and $((i_1/o_1, \ldots, i_r/o_r), \hat{t'})$ belong to $\mathtt{TEvol}(S)$, then

$$I\ \mathtt{conf}_{w}^{int}\ S \text{ iff } I\ \mathtt{conf}_{b}^{int}\ S$$
$$I\ \mathtt{conf}_{lp}^{int}\ S \text{ iff } I\ \mathtt{conf}_{mp}^{int}\ S$$
$$I\ \mathtt{conf}_{sw}^{int}\ S \text{ iff } I\ \mathtt{conf}_{sb}^{int}\ S$$
$$I\ \mathtt{conf}_{slp}^{int}\ S \text{ iff } I\ \mathtt{conf}_{smp}^{int}\ S$$

(2) If for all non-temporal evolution $(i_1/o_1, \ldots, i_r/o_r) \in \mathtt{NTEvol}(I)$ there do not exist two different intervals $\hat{t}, \hat{t'} \in \mathcal{I}_{\mathbb{R}_+}$ such that $((i_1/o_1, \ldots, i_r/o_r), \hat{t})$ and $((i_1/o_1, \ldots, i_r/o_r), \hat{t'})$ belong to $\mathtt{TEvol}(I)$, then

$$I\ \mathtt{conf}_{w}^{int}\ S \text{ iff } I\ \mathtt{conf}_{sw}^{int}\ S$$
$$I\ \mathtt{conf}_{lp}^{int}\ S \text{ iff } I\ \mathtt{conf}_{slp}^{int}\ S$$
$$I\ \mathtt{conf}_{b}^{int}\ S \text{ iff } I\ \mathtt{conf}_{sb}^{int}\ S$$
$$I\ \mathtt{conf}_{mp}^{int}\ S \text{ iff } I\ \mathtt{conf}_{smp}^{int}\ S$$

(3) If the conditions of the results (1) and (2) hold then the relations $\mathtt{conf}_{w}^{int}$, $\mathtt{conf}_{b}^{int}$, $\mathtt{conf}_{sw}^{int}$, and $\mathtt{conf}_{sb}^{int}$ coincide and the relations $\mathtt{conf}_{lp}^{int}$, $\mathtt{conf}_{mp}^{int}$, $\mathtt{conf}_{slp}^{int}$, and $\mathtt{conf}_{smp}^{int}$ coincide too.

□

Now, we will define implementation relations in a black box testing framework, such as we have been considering along the paper. In this case, we have no information about the time intervals that appear in the implementation. As we did

in Section 4.3, we will base our relations on a set of observed timed executions. We will establish two levels of conformance. In the *global* relations, for each evolution e, we will evaluate the agreement of the interval containing the time values that have been collected in the sample (that is, $\widehat{S}_{(H,e)}$) with respect to the *global behavior* required for it in the model. This global behavior will be defined by means of the *coverage interval* of each evolution, in which we integrate all possible time intervals associated with the corresponding timed evolutions in the model. The second level of relations that we will establish, *sometimes* relations, requires for each evolution e represented in the sample, that there exists a timed evolution $(e, \hat{t})$ in the model such that the time values in the sample associated to e satisfy the appropriate conditions with respect to $\hat{t}$.

Definition 24 Let I and S be two TFSMs. Let H be a multiset of observed timed executions of I and $\Phi = \{e \mid \exists\ t : (e,t) \in H\} \cap \mathtt{NTEvol}(S)$. Given $e \in \Phi$, we define the *coverage interval* of e in S as the interval $\widehat{Cov}_{(S,e)} = [t_1, t_2]$ where

$$t_1 = \mathtt{min}\{\pi_1(\hat{t}) \mid (e,\hat{t}) \in \mathtt{TEvol}(S)\}$$
$$t_2 = \mathtt{max}\{\pi_2(\hat{t}) \mid (e,\hat{t}) \in \mathtt{TEvol}(S)\}$$

We define the following implementation relations:

- I $H-$*in time globally conforms* to S, denoted by $I\ \mathtt{confg}^H_{int}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all evolution $e \in \Phi$ we have that $\widehat{S}_{(H,e)} \subseteq \widehat{Cov}_{(S,e)}$.
- I $H-$*fast globally conforms* to S, denoted by $I\ \mathtt{confg}^H_{intf}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all evolution $e \in \Phi$ we have that $\widehat{S}_{(H,e)} \ll \widehat{Cov}_{(S,e)}$.
- I $H-$*preferable globally conforms* to S, denoted by $I\ \mathtt{confg}^H_{intp}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all evolution $e \in \Phi$ we have that $\widehat{S}_{(H,e)} \preceq \widehat{Cov}_{(S,e)}$.
- I $H-$*sometimes in time conforms* to S, denoted by $I\ \mathtt{confs}^H_{int}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all evolution $e \in \Phi$ there exists $(e,\hat{t}) \in \mathtt{TEvol}(S)$ such that $\widehat{S}_{(H,e)} \subseteq \hat{t}$.
- I $H-$*sometimes fast conforms* to S, denoted by $I\ \mathtt{confs}^H_{intf}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all evolution $e \in \Phi$ there exists $(e,\hat{t}) \in \mathtt{TEvol}(S)$ such that $\widehat{S}_{(H,e)} \ll \hat{t}$.
- I $H-$*sometimes preferable conforms* to S, denoted by $I\ \mathtt{confs}^H_{intp}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all evolution $e \in \Phi$ there exists $(e,\hat{t}) \in \mathtt{TEvol}(S)$ such that $\widehat{S}_{(H,e)} \preceq \hat{t}$.

□

7.3. Definition, application, and derivation of tests

Next, we present the algorithm we use for deriving tests from a non-deterministic TFSM. This algorithm is an extension of the one presented in Section 6. Due to the fact that in this section we allow TFSMs to present fully non-deterministic behaviors, we may have several timed evolutions corresponding to the same non-timed evolution. So, when deriving tests, we need to include all the possible time values for each possible evolution.

The only difference with respect to the notion of test introduced in Definition 16 concerns the C_T function that associates time values with passing states. In this case, the range of the function will be a subset of time values (either real numbers or time intervals) instead of a single time value. That is, we will consider functions $C_T : S_P \longrightarrow \mathcal{P}(\mathbb{R}_+)$ for time requirements expressed by using fix time values and functions $C_T : S_P \longrightarrow \mathcal{P}(\mathcal{I}_{\mathbb{R}_+})$ for time requirements expressed by time intervals.

Next, we consider the application of tests to implementations by taking into account the different time conditions. The aspect relating to pass a test suite without consider time requirements coincides with the one presented in Definition 17. In the next definition we present the notion of passing a test regarding time expressed by means of fix time values.

Definition 25 Let I be a TFSM, T be a test, and s^T be a state of T. We say that

- I *passes* the test suite $\mathcal{T}_{st}$ *for all time* if $\mathtt{pass}(I, \mathcal{T}_{st})$ and for all $\sigma = (e,t) \in \mathtt{TEvol}(I)$ and all $(T, s^T) \in \mathtt{Test}(\sigma, \mathcal{T}_{st})$, we have that $t \in C(s^T)$ holds.

- *I passes* the test suite $\mathcal{T}_{st}$ *in the best time* if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $\sigma = (e,t) \in \texttt{TEvol}(I)$, all $(T, s^T) \in \texttt{Test}(\sigma, \mathcal{T}_{st})$, and all $t_c \in C(s^T)$, we have that $t \leq t_c$ holds.
- *I passes* the test suite $\mathcal{T}_{st}$ *in the worst time* if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $\sigma = (e,t) \in \texttt{TEvol}(I)$ and all $(T, s^T) \in \texttt{Test}(\sigma, \mathcal{T}_{st})$, there exists $t_c \in C(s^T)$ such that $t \leq t_c$ holds.
- *I passes* the test suite $\mathcal{T}_{st}$ *sometimes in best time* if $\texttt{pass}(I, \mathcal{T}_{st})$ and there exists $\sigma = (e,t) \in \texttt{TEvol}(I)$ such that for all $(T, s^T) \in \texttt{Test}(\sigma, \mathcal{T}_{st})$ and all $t_c \in C(s^T)$, we have that $t \leq t_c$ holds.
- *I passes* the test suite $\mathcal{T}_{st}$ *sometimes in worst time* if $\texttt{pass}(I, \mathcal{T}_{st})$ and there exist $\sigma = (e,t) \in \texttt{TEvol}(I)$, $(T, s^T) \in \texttt{Test}(\sigma, \mathcal{T}_{st})$, and $t_c \in C(s^T)$ such that $t \leq t_c$ holds.

□

Concerning the notions of passing test suites when time requirements are expressed by using time intervals, we will consider only a black box testing framework. First, we need to adapt the concept of coverage interval to deal with time intervals appearing in tests.

Definition 26 Let us consider a test $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C_T)$. Given a passing state $s^T \in S_P$, we define the *coverage interval* for the state s^T as the interval $\widehat{Cov}_{(T,s^T)} = [t_1, t_2]$ where

$$t_1 = \texttt{min}\{\pi_1(\hat{t}) \mid \hat{t} \in C_T(s^T)\}$$
$$t_2 = \texttt{max}\{\pi_2(\hat{t}) \mid \hat{t} \in C_T(s^T)\}$$

Let I be a TFSM and $\mathcal{T}_{st} = \{T_1, \ldots, T_n\}$ be a test suite. Let $H_1, \ldots, H_n$ be test execution samples of I and $T_1, \ldots, T_n$, respectively. Let $H = \bigcup_{i=1}^{n} H_i$ and $\Phi = \{e \mid \exists\ t : (e,t) \in H\}$. We say that

- *I H-passes in time globally* the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, we have $\widehat{S}_{(H,e)} \subseteq \widehat{Cov}_{(T,s^T)}$.
- *I H-passes fast globally* the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, we have $\widehat{S}_{(H,e)} \ll \widehat{Cov}_{(T,s^T)}$.
- *I H-passes preferable globally* the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, we have $\widehat{S}_{(H,e)} \preceq \widehat{Cov}_{(T,s^T)}$.
- *I H-passes sometimes in time* the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, there exists $\hat{t} \in C_T(s^T)$ such that $\widehat{S}_{(H,e)} \subseteq \hat{t}$.
- *I H-passes sometimes fast* the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, there exists $\hat{t} \in C_T(s^T)$ such that $\widehat{S}_{(H,e)} \ll \hat{t}$.
- *I H-passes sometimes preferable* the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}_{st}$ such that $I \| T \stackrel{e}{\Longrightarrow} s^T$, there exists $\hat{t} \in C_T(s^T)$ such that $\widehat{S}_{(H,e)} \preceq \hat{t}$.

□

Next we give the extension of the algorithm presented in Figure 6 to deal with nondeterminism. First, we need to adapt the notion of after, introduced in Definition 21, to the new setting. Essentially, after will return a set of pairs instead of a single element. The definition of out remains the same as in Definition 21.

Definition 27 Let $M = (S, I, O, Tr, s_{in})$ be a TFSM. We define the function $\texttt{after} : S \times I \times O \times \mathcal{T}\texttt{ime} \longrightarrow \mathcal{P}(S \times \mathcal{T}\texttt{ime})$ such that for all $s \in S, i \in I, o \in O$, and $d \in \mathcal{T}\texttt{ime}$ we have

$$\texttt{after}(s, i, o, d) = \{(s', d+d') | \exists\ (s, s', i, o, d') \in Tr\}$$

□

In contrast with the definition of after for observable machines, the function $\texttt{after}(s, i, o, d)$ now returns a set of *situations* that can be reached from a state s after receiving the input i, producing the output o, when the duration of the process is given by d.

The `out` and `after` functions can be extended in the natural way to deal with sets:

$$\begin{aligned} \texttt{out}(C_o, i) &= \bigcup_{s \in C_o} \texttt{out}(s, i) \\ \texttt{after}(D, i, o) &= \bigcup_{(s,d) \in D} \texttt{after}(s, i, o, d) \end{aligned}$$

The algorithm to derive tests from a `TFSM` is similar to the one presented in Section 6. Next we show the differences with respect to that one. The first inductive case becomes:
"If $(D, s^T) \in S_{aux}$ then perform the following steps: $S_{aux} := S_{aux} - \{(D, s^T)\}; S_P := S_P \cup \{s^T\}$; $C(s^T) := \{d \mid (s, d) \in D\}$"
The condition in the second inductive case becomes:
"If $S_{aux} = \{(D, s^T)\}$ and $\exists\, i \in I : \texttt{out}(S_M, i) \neq \emptyset$, with $S_M = \{s \mid (s, d) \in D\}$"
Finally, the third and fourth items of 2(f) should be replaced by the assignments "$D' := \texttt{after}(D, i, o)$" and "$S_{aux} := S_{aux} \cup \{(D', s'')\}$", respectively.

Regarding the behavior of the algorithm, a set of *pending situations* D keeps those pairs denoting the possible states and duration values that could appear in a state of the test whose definition, that is, its outgoing transitions, has not been yet completed. A pair $(D, s^T) \in S_{aux}$ indicates that we did not complete the state s^T of the test and that the possible situations for that state are given by the set D. Let us remark that D is a set of situations, instead of a single one, due to the non-determinism that can appear in the corresponding `TFSM`.

Example 4 Let $M = (S, I, O, Tr, s_{in})$ be a `TFSM`. Let us suppose that we have two transitions $(s, s', i, o, d_1), (s, s'', i, o, d_2) \in Tr$. In order to compute the evolutions of M after performing i/o we have to consider both s' and s''. Formally, for a configuration $(s, \bar{x})$ and taking into account that the time elapsed so far equals d, we have to consider the set $\texttt{after}(\{(s, d)\}, i, o) = \{(s', d_1), (s'', d_2)\}$. Thus, the application of this function will return the different configurations, as well as the total durations, that could be obtained from s and duration d after receiving the input i and generating the output o. □

In order to show that the test suites obtained by applying the previous algorithm are sound and complete with respect to the implementation relations defined in Section 7.1 and 7.2, we will follow a reasoning similar to that in Theorems 1 and 3. In the case of systems whose time requirements are expressed by means of fix time values, the main difference with respect to the proof given in Theorem 1 is that *pending situations* may contain a set of possibilities instead of a single one. In addition, we have to take into account that the adaptation to the new non-deterministic setting of the algorithm presented in Figure 6 associates a set of time values to passing states instead of a single value. Thus, we only need to change the points of the proof where we refer to it. In this regard, the proof remains as we developed in Theorem 1, except that expressions such as $t \neq C_T(s_T)$ must by replaced by expressions such as $t \notin C_T(s_T)$.

Theorem 6 Let S and I be two `TFSM`s. We have that:

- $I\ \texttt{conf}_a\ S$ iff I passes $\texttt{tests}(S)$ for all time.
- $I\ \texttt{conf}_b\ S$ iff I passes $\texttt{tests}(S)$ in the best time.
- $I\ \texttt{conf}_w\ S$ iff I passes $\texttt{tests}(S)$ in the worst time.
- $I\ \texttt{conf}_{sb}\ S$ iff I passes $\texttt{tests}(S)$ in sometimes in best time.
- $I\ \texttt{conf}_{sw}\ S$ iff I passes $\texttt{tests}(S)$ in sometimes in worst time.

□

Next, we present a result to establish the relation between passing the test suite $\texttt{tests}(S)$ for a sample H, and the global implementation relations ($\texttt{confg}^H_{int}$, $\texttt{confg}^H_{intf}$, and $\texttt{confg}^H_{intp}$) and the sometimes implementation relations ($\texttt{confs}^H_{int}$, $\texttt{confs}^H_{intf}$, and $\texttt{confs}^H_{intp}$) given in Definition 24.

Theorem 7 Let I and S be two `TFSM`s. Given a multiset of timed executions H we have:

- $I\ \texttt{confg}^H_{int}\ S$ iff I *$H-$passes in time globally* $\texttt{tests}(S)$.

- I $\mathtt{confg}^H_{intf}$ S iff I $H-$*passes fast globally* $\mathtt{tests}(S)$.
- I $\mathtt{confg}^H_{intp}$ S iff I $H-$*passes preferable globally* $\mathtt{tests}(S)$.
- I $\mathtt{confs}^H_{int}$ S iff I $H-$*passes sometimes in time* $\mathtt{tests}(S)$.
- I $\mathtt{confs}^H_{intf}$ S iff I $H-$*passes sometimes fast* $\mathtt{tests}(S)$.
- I $\mathtt{confs}^H_{intp}$ S iff I $H-$*passes sometimes preferable* $\mathtt{tests}(S)$.

Proof Sketch: $I\,\mathtt{confg}^H_{int}\,S$ requires $I\,\mathtt{conf}_{nt}\,S$ and that for all evolution e belonging to the set of observed timed executions, the sample interval of e in H belongs to the coverage interval of e in the specification. In the same way that we have followed for fix time values, the non-temporal aspect of the proof coincides with the proof of Theorem 1. Regarding temporal conditions, the samples collected from the tests applications are exactly the ones we apply to check whether the implementation relation holds. Given an evolution e appearing in the sample H, the implementation relation demands that the interval $\widehat{S}_{(H,e)}$ is included in the coverage interval of e in S, that is, $\widehat{Cov}_{(S,e)}$. Applying our algorithm, each derived test T associates to each passing state all the possible time intervals that the specification may present to perform the evolution e. So, for all test T such that $T \stackrel{e}{\Longrightarrow} s^T$ we have that $\widehat{Cov}_{(S,e)} = \widehat{Cov}_{(T,s^T)}$, being the requirements established to pass the tests the same as the ones demanded by the implementation relation. □

8. Related work

In this section we review some of the work most related to ours. A longer discussion on related work can be found in [23].

Due to the intrinsical difficulty of testing timed systems, different approaches have been studied, falling each of them into one or more of the following categories:

(a) Uncomplete testing;

(b) complete finite testing; and;

(c) complete infinite testing.

In the first category only some behaviors, out of those that are relevant for the correctness of the implementation, are tested (see e.g. [7,6,4,13,19]). In these cases, methods to choose those tests that seem to have a higher capability to find errors are proposed. There are two possible approaches for doing this. On the one hand, we may consider a given *coverage criterium* to build our tests [7,6,19]. In this case, a test suite is constructed in such a way that its tests would fully traverse a given structural characteristic of the IUT (e.g., *states*, *locations*, *edges*, etc) if the IUT were correct indeed. By doing this, we expect the coverage of this characteristic in the actual IUT to be high. On the other hand, we may restrict the behavior to be tested to some *test purposes* denoting critical scenarios [13], or we may split the specification into different *test views* [4] and individually test each of them as a way to avoid the state explosion (with the risk of missing faults that are caused by the relations between test views).

In the (b) category, a *complete finite* test suite is derived from the specification, that is, if all tests in the finite suite are passed then the implementation is correct (see e.g. [5,28]). Usually, the finiteness of this suite requires to introduce strong assumptions about the implementation, both to deal with functional requirements (e.g. to assume that the maximal number of states in the implementation is known) and timed requirements (e.g. urgency of outputs or discretization of time). In general, the applicability of the derived test suite is not feasible because the number of derived tests is astronomic. Let us note that *uncomplete* methods, like those considered in the previous category, may also seek for completeness when testing a *specific* test purpose (i.e., they could try to exhaustively test *all* behaviors concerned by the purpose). In this case, the same kind of strong assumptions are required.

Finally, in the (c) category a complete *infinite* test suite is extracted from the specification (see e.g. [22,27,24,3]). In particular, a test derivation algorithm is defined in such a way that, for all implementation behaviors that must be tested be-

fore granting correctness, a suitable test for checking this behavior is added by the algorithm. In this sense, such an algorithm is complete (that is, it provides full fault coverage with respect to the considered testing relation) *in the limit*. The interest of these methods is that, on the one hand, weaker assumptions are required in these methodologies and, on the other hand, being provided with a method to find and construct any required test is needed if we want to *select* some of these tests according to some criteria. That is, methods that are exhaustive in the limit are the basis for other non-exhaustive, but more practical, methods. The methodology presented in this paper fits into this category and, consequently, its aim is to provide test suites that are complete in the limit while, in turn, no strong assumptions are required (e.g. about the number of states of the implementation).

Alternatively, we may classify timed testing methods in terms of the underlying timed model used to denote specifications and implementations. Several approaches use *timed automata* [1] or variants where some restrictions are introduced to make testing a feasible task (see e.g. [28,16, 4]). Other works use *clock region graphs* [13] or temporal extensions of *labeled transition systems* (e.g. [5,19,3]), *extended finite state machines* [17], or *process algebra* [7,6]. In technical terms, the selection of the underlying model depends on the kind of system assumptions we wish to consider (for instance, whether we need to keep time values of past transitions, using variables to affect actual time values, linking inputs and outputs, assuming urgency of outputs or discretization of time, etc). Besides, some timed testing methods are timed adaptations of other previous untimed methods. For instance, [17] presents an adaptation of the UIOv-method [32], algorithms in [11,10,18] are extensions of the Wp-method [14] to timed systems, while [19,26,3] are based on the non-deterministic test derivation algorithm for `ioco` constructed in [29,30]. This algorithm is also the basis of the timed test derivation algorithm presented in this paper.

Compared to other timed testing approaches, this paper presents a unified framework where deterministic time, time intervals, and stochastic time are treated. Let us emphasize that time is usually represented in deterministic terms in temporal testing methods. Only few approaches represent time as intervals (e.g. [7,6,5]). Regarding stochastic time, to the best of our knowledge only our previous work presented in [25], commented before, provides a method to *black-box test* implementations having stochastic time.

9. Concluding remarks

In this paper we have presented a formal model to test systems where temporal requirements are very relevant. The main contribution of this novel framework is the integration of different time domains in a single methodology. Specifically, we use a uniform formalism to describe systems where time requirements can be expressed either by using fix time values, random variables, or time intervals. This formalism is an extension of *Finite State Machines*. We have defined a common conformance relation for the non-functional requirements of the systems. This relation is taken as the first step to define several relations considering time constraints. These relations take into account the special features of each time domain that we have studied. While implementation relations using fix time values are relatively standard, if time conditions are expressed by means of random variables or time intervals we need to apply a method based on a set of observations obtained from the interaction with the implementation. In the case of random variables we applied a hypothesis contrast for determining the similarity level of the random variable extracted from the model and the observed time values. The criterion followed for systems where time is expressed by means of time intervals is to check that the observed time values in the sample belong to the time interval of the specification. In order to ease the presentation, we initially restricted the exposition to observable machines. However, non-determinism is an important feature while describing systems. Thus, we have extended the timed conformance relations to deal with systems that present non-deterministic behaviors.

In addition to study implementation relations,

we have introduced a common formal testing framework. We have given a general definition of test that includes time conditions independently of the time domain under consideration. We have also given an algorithm to derive a sound and complete test suite from a given formal model which allows us to check if an implementation fulfills, with respect to the model, the timed conformance relations previously defined. Following the same pattern, an algorithm for non-deterministic behaviors has been developed considering the set of time values that each evolution could take in its performance. This is so because non-determinism induces that the same evolution can be performed by taking different time amounts.

Acknowledgements

We would like to thank the anonymous reviewers of this paper for the careful reading and the suggestions that have improved the quality of the paper. We would also like to thank the responsible editor, Reinhard Gotzhein, for his useful comments and his guidance during the reviewing process of the paper.

10. Appendix - Statistics background: Hypothesis contrasts.

In this appendix we introduce one of the standard ways to measure the confidence degree that a random variable has on a sample. In order to do so, we will present a methodology to perform *hypothesis contrasts*. The underlying idea is that a sample will be *rejected* if the probability of observing that sample from a given random variable is low. In practice, we will check whether the probability to observe a *discrepancy* lower than or equal to the one we have observed is low enough. We will present *Pearson's* χ^2 *contrast*. This contrast can be applied both to continuous and discrete random variables. The mechanism is the following. Once we have collected a sample of size n we perform the following steps:

- We split the sample into k classes which cover all the possible range of values. We denote by O_i the *observed frequency* at class i (i.e. the number of elements belonging to the class i).
- We calculate the probability p_i of each class, according to the proposed random variable. We denote by E_i the *expected frequency*, which is given by $E_i = np_i$.
- We calculate the *discrepancy* between observed frequencies and expected frequencies as $X^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$. When the model is correct, this discrepancy is approximately distributed as a random variable χ^2 .
- We estimate the number of freedom degrees of χ^2 as $k - r - 1$. In this case, r is the number of parameters of the model which have been estimated by maximal likelihood over the sample to estimate the values of p_i (i.e. $r = 0$ if the model completely specifies the values of p_i before the samples are observed).
- We will *accept* that the sample follows the proposed random variable if the probability to obtain a discrepancy greater or equal to the discrepancy observed is high enough, that is, if $X^2 < \chi^2_\alpha(k - r - 1)$ for some α high enough. Actually, as such margin to accept the sample decreases as α increases, we can obtain a measure of the validity of the sample as $\texttt{max}\{\alpha \mid X^2 < \chi^2_\alpha(k - r - 1)\}$.

According to the previous steps, we can now present an operative definition of the function γ which is used in this paper to compute the confidence of a random variable on a sample.

Definition 28 Let ξ be a random variable and J be a multiset of real numbers representing a sample. Let X^2 be the discrepancy level of J on ξ calculated as explained above by splitting the sampling space into k classes

$$C = \{[0, a_1), [a_1, a_2), \ldots, [a_{k-2}, a_{k-1}), [a_{k-1}, \infty)\}$$

where k is a given constant and for all $1 \leq i \leq k - 1$ we have $P(\xi \leq a_i) = \frac{i}{k}$. We define the confidence of ξ on J with classes C, denoted by $\gamma(\xi, J)$, as $\texttt{max}\{\alpha \mid X^2 < \chi^2_\alpha(k - 1)\}$. □

The previous definition indicates that in order to perform a contrast hypothesis, we split the collected values in several intervals having the same expected probability. We compute the value for X^2 as previously described and check this figure with the tabulated tables (see, for example, `www.statsoft.com/textbook/sttable.html`) corresponding to χ^2 with $k-1$ freedom degrees.

Let us comment some important details. First, given the fact that the random variables that we use in our framework denote the passing of time, we do not need classes to cover negative values. Thus, we will suppose that the class containing 0 will also contain all the negative values. Second, let us remark that in order to apply this contrast it is strongly recommended that the sample has at least 30 elements while each class must contain at least 3 elements.

Example 5 Let us consider a device that produces real numbers belonging to the interval $[0, 1]$. We would like to test whether the device produces these numbers randomly, that is, it does not have a number or sets of numbers that are more probable to be produced than other ones. Thus, we obtain a sample from the machine and we apply the contrast hypothesis to determine whether the machine follows a uniform distribution in the interval $[0, 1]$. First, we have to decide how many classes we will use. Let us suppose that we take $k = 10$ classes. Thus, for all $1 \leq i \leq 9$ we have $a_i = 0.i$ and $P(\xi \leq a_i) = \frac{i}{10}$. So, $C = \{[0, 0.1), [0.1, 0.2) \ldots [0.8, 0.9), [0.9, \infty)\}$.

Let us suppose that the multiset of observed values, after we sort them, is:

$$J = \left\{\begin{array}{l} 0.00001, 0.002, 0.0876, 0.8, \\ 0.1, 0.11, 0.123, \\ 0.21, 0.22, 0.22, 0.2228, 0.23, 0.24, 0.28, \\ 0.32, 0.388, 0.389, 0.391 \\ 0.4, 0.41, 0.42, 0.4333 \\ 0.543, 0.55, 0.57, \\ 0.62, 0.643, 0.65, 0.67, 0.68, 0.689, 0.694 \\ 0.71, 0.711, 0.743, 0.756, 0.78, 0.788, \\ 0.81, 0.811, 0.82, 0.845, 0.8999992, \\ 0.91, 0.93, 0.94, 0.945, 0.9998 \end{array}\right\}$$

Since the sample has 48 elements we have that the expected frequency in each class, E_i, is equal to 4.8. In contrast, the observed frequencies, O_i, are $4, 3, 7, 4, 4, 3, 7, 6, 5, 5$. Next, we have to compute

$$X^2 = \sum_{i=1}^{10} \frac{(O_i - E_i)^2}{E_i} = 4.08333$$

Finally, we have to consider the table corresponding to χ^2 with 9 freedom degrees and find the maximum α such that $4.08333 < \chi^2_\alpha(9)$. Since $\chi^2_{0.9}(9) = 4.16816$ and $\chi^2_{0.95}(9) = 3.32511$ we conclude that, with probability 0.9, the machine produces indeed random values. □

REFERENCES

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. M. Bernardo and W.R. Cleaveland. A theory of testing for markovian processes. In *11th Int. Conf. on Concurrency Theory, CONCUR'2000, LNCS 1877*, pages 305–319. Springer, 2000.
3. L. Brandán Briones and E. Brinksma. Testing real-time multi input-output systems. In *7th Int. Conf. on Formal Engineering Methods, ICFEM'05, LNCS 3785*, pages 264–279. Springer, 2005.
4. R. Cardell-Oliver. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.
5. R. Cardell-Oliver and T. Glover. A practical and complete algorithm for testing real-time systems. In *5th Int. Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'98, LNCS 1486*, pages 251–260. Springer, 1998.
6. D. Clarke, Y.S. Kim, and I. Lee. Automatic test generation for the analysis of a real-time system: Case study. In *3rd IEEE Real Time Technology and Applications Symposium, RTAS'97*, pages 112–124. IEEE Computer Society Press, 1997.
7. D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *3rd Workshop on Object-Oriented*

Real-Time Dependable Systems, WORDS'97, pages 199–206. IEEE Computer Society Press, 1997.
8. R. de Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
9. A. En-Nouaary and R. Dssouli. A guided method for testing timed input output automata. In *15th Int. Conf. on Testing Communicating Systems, TestCom'03, LNCS 2644*, pages 211–225. Springer, 2003.
10. A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-method: Testing real time systems. *IEEE Transactions on Software Engineering*, 28(11):1024–1039, 2002.
11. A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi. Timed test cases generation based on state characterization technique. In *19th IEEE Real Time Systems Symposium, RTSS'98*, pages 220–231. IEEE Computer Society Press, 1998.
12. M.A. Fecko, M.Ü. Uyar, A.Y. Duale, and P.D. Amer. A technique to generate feasible tests for communications systems with multiple timers. *IEEE/ACM Transactions on Networking*, 11(5):796–809, 2003.
13. H. Fouchal, E. Petitjean, and S. Salva. An user-oriented testing of real time systems. In *IEEE Workshop on Real-Time Embedded Systems, RTES'01*. IEEE Computer Society Press, 2001.
14. S. Fujiwara, G. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite-state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991.
15. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
16. A. Hessel, K. Larsen, B. Nielsen, P. Petterson, and A. Skou. Time-optimal real-time test case generation using UPPAAL. In *3rd Int. Workshop on Formal Approaches to Testing of Software, FATES'03, LNCS 2931*, pages 114–130. Springer, 2003.
17. T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed I/O automaton model. In *12th Int. Workshop on Testing of Communicating Systems, IWTCS'99*, pages 197–214. Kluwer Academic Publishers, 1999.
18. A. Khoumsi. A method for testing the conformance of real-time systems. In *7th Int. Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'02, LNCS 2469*, pages 331–354. Springer, 2002.
19. M. Krichen and S. Tripakis. An expressive and implementable formal framework for testing real-time systems. In *17th Int. Conf. on Testing of Communicating Systems, TestCom'05, LNCS 3502*, pages 209–225. Springer, 2005.
20. K.G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using Uppaal. In *4th Int. Workshop on Formal Approaches to Testing of Software, FATES'04, LNCS 3395*, pages 79–94. Springer, 2004.
21. N. López and M. Núñez. A testing theory for generally distributed stochastic processes. In *12th Int. Conf. on Concurrency Theory, CONCUR'01, LNCS 2154*, pages 321–335. Springer, 2001.
22. D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real time systems from logic specifications. *ACM Transactions on Computer Systems*, 13(4):356–398, 1995.
23. M.G. Merayo. An integrated framework for the study of conformance relations in timed systems. Master's thesis, Universidad Complutense de Madrid, 2006. Available at `http://kimba.mat.ucm.es/testing/papers/master-merayo.pdf`.
24. M. Núñez and I. Rodríguez. Encoding `PAMR` into (timed) `EFSMs`. In *22nd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'02, LNCS 2529*, pages 1–16. Springer, 2002.
25. M. Núñez and I. Rodríguez. Towards testing stochastic timed systems. In *23rd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'03, LNCS 2767*, pages 335–350. Springer, 2003.
26. M. Núñez and I. Rodríguez. Conformance testing relations for timed systems. In *5th Int. Workshop on Formal Approaches to Software Testing, FATES'05, LNCS 3997*, pages 103–117. Springer, 2006.

27. J. Peleska and M. Siegel. Test automation of safety-critical reactive systems. *South African Computer Journal*, 19:53–77, 1997.
28. J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001. Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.
29. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.
30. J. Tretmans. Testing concurrent systems: A formal approach. In *10th Int. Conf. on Concurrency Theory, CONCUR'99, LNCS 1664*, pages 46–65. Springer, 1999.
31. M.Ü. Uyar, M.A. Fecko, A.S. Sethi, and P.D. Amar. Testing protocols modeled as FSMs with timing parameters. *Computer Networks*, 31(18):1967–1998, 1999.
32. S.T. Voung, W.L. Chan, and M.R. Ito. The UIOv-method for protocol test sequence generation. In *2nd IFIP TC6 Int. Workshop on Protocol Test Systems, IWPTS'89*, pages 161–175. North-Holland, 1990.