Extending EFSMs to specify and test timed systems with action durations and timeouts

Mercedes G. Merayo, Manuel Núñez and Ismael Rodríguez

Abstract— **In this paper we introduce a timed extension of the extended finite state machines model. On the one hand, we consider that (output) actions take time to be performed. This time may depend on several factors such as the value of variables. On the other hand, our formalism allows to specify timeouts. In addition to presenting our language, we develop a testing theory. First, we define ten timed conformance relations and relate them. Second, we introduce a notion of timed test and define how to apply tests to implementations. Finally, we give an algorithm to derive sound and complete test suites with respect to the implementation relations presented in the paper.**

Index Terms— **D.2.5 Testing and Debugging; D.2.4.d Formal methods;F.3.1.f Specification techniques; F.3.1 Specifying and Verifying and Reasoning about Programs;**

I. Introduction

Real-time systems are hardware and software systems which are subject to timing constraints, that is, the time that the system can spend for producing a response has a deadline. These systems can be found in several branches of the industrial world, for example, in aeronautics and medicine. It would be unacceptable that a system providing a medical treatment, for example radiotherapy, does not consider critical situations, allowing that an extra dose of radiation would be applied to a patient. It would be desirable that the system could react to these situations autonomously, stopping the treatment if the system does not stop before a bounded amount of time. Another practical example can be found in the security in public transport. Usually, trains decrease the speed uniformly if the driver does not push a button with a preestablished frequency. This behaviour avoids the train to cause an accident if the driver loses consciousness. Thus, the necessity to assure the correctness of real-time systems forces to apply a methodology which considers time requirements. In most models, temporal requirements usually refer to time that a system consumes for performing operations. However, another kind of time constraints can affect systems: *Timeouts*. A timeout is a specified period of time that will be allowed to elapse in a system waiting for an interaction with it; if the period ends and no action has been produced, the state of the system changes and its reactions to the actions that are received from the environment may be different.

Formal methods allow the analysis of systems and the reasoning about them with mathematical precision and rigor. It is widely recognized that formal methods and testing are complementary techniques ([2], [3], [4], [5], [6], [7]), since they help to check the correctness of systems and provide a framework for testing. The use of formal methods for describing a specification provides a more precise and consistent starting point for the testing process. Hence, it is necessary to provide languages to formally specify these systems. Even though there exists a myriad of timed extensions of classical frameworks [8], [9], [10], [11], [12], [13], [14], [15], [16], most of them specialize only in one of the previous variants: Time is either associated with actions or associated with delays/timeouts. Our formalism based on *extended finite state machines*, allows to specify in a natural way both time aspects.

The theoretical framework is complicated by two additional features. First, we consider that variables may influence the timing aspects of the specification. Thus, the execution of an action may take different time values if the value of the variables change. Second, we do not impose any restriction on the deterministic behavior of our machines. This implies, again, that the same sequence of actions may take different time values to be executed.

We assume that both, specifications and implementations, can be modelled by means of the previous formalism, and propose a formal testing methodology to systematically test a system with respect to a specification. Testing a system requires using a specific notion of correctness. When we deal with real-time systems it is necessary to establish two different levels of conformance: *Functional and timed conformance*. Regarding functional conformance we have to consider that the sequences of inputs/outpus produced by the system under test must be allowed in the specification, that is, the implementation does not *invent* anything for those inputs that are *specified* in the specification. In addition, we have to take into account the possible timeouts. For example, a sequence of inputs/outputs could be accepted only after different timeouts have been triggered.

Regarding temporal performance requirements, our testing methodology will take into account only the consumed time while performing actions, not the time that the system waits for a reaction from the environment. The latter solely affects which actions are allowed/forbidden.

We propose several timed conformance relations according to the interpretation of *good* implementation for a given specification. Let us emphasize that testing the temporal behavior of a system affected by non-determinism requires to face some issues that are not considered by other testing approaches. For instance, we may require that among all the time values the implementation may need to perform a task, one of them is

This paper represents an extended and improved version of [1]. Research partially supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01) and the Marie Curie project MRTN-CT-2003-505121/TAROT.

The authors are with Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, 28040 Madrid, Spain. e-mail: mgmerayo@fdi.ucm.es, {mn,isrodrig}@sip.ucm.es

smaller than the one corresponding to the specification for this task. Hence, if the application of a test to the implementation takes a long time, this does not necessarily mean that it cannot do it faster.

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

In terms of related work, our language is based on *extended finite state machines*, which have been extensively used by the formal testing community. This paper continues the work in [17]. The main advantage with respect to this previous work is that we can now express timeouts, we remove all the restrictions regarding non-determinism of the machines, and we consider more conformance relations. Our way to deal with time is completely different to that in *timed automata* [13]. As we said before, we can associate time with the performance of actions while timeouts can be easily represented. These features do not only improve the modularity of models, but they are also suitable for clearly identifying system requirements and responsibilities in a testing methodology. Our model implies the existence of a unique clock to control how time passes. This fact can make our model to be apparently less expressive than timed automata, where different clocks can be used. For instance, it is not direct to express time relations between different parts of the machines (for example, to specify that a transition takes 2 time units more than the previously performed transition). However, these situations can be easily simulated by using additional variables (we are working within an EFSM model where variables, associated with different domains, can be used) to record relevant time values. However, it is worth to mention that our notion of timeout is related to having invariants associated with states of a timed automata as described in [18]: Once the value of a clock variable exceeds the invariant, the system produces a prescribed output and enters a new state. Among those models based on timed automata, the ones most related to ours are those where urgency has been added to the performance of transitions (e.g., [19], [20], [21]).

Regarding testing of temporal requirements, there exist several proposals (e.g., [22], [23], [18], [24], [25], [26]) but most of them are based on timed automata. Moreover, in these approaches tests are independent and the diagnosis of a test does not depend on other tests. By considering that tests are interrelated, we can relate non-determinism and temporal requirements, as well as define and apply several conformance relations where non-determinism is explicitly considered.

Due to the intrinsical difficulty behind testing timed systems, different approaches have been studied, falling each of them into one or more of the following categories: (a) Only some behaviors, out of those that are relevant for the correctness of the implementation, are tested (see e.g. [27], [28]). In these cases, methods to choose those tests that seem to have a higher capability to find errors are proposed, though they are usually heuristic or are based on restricting the behavior to be tested to some specific *test purposes*; (b) 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. [29], [18]). 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, discretization of time). In general, the applicability of the derived test suite is not feasible because the number of derived tests is astronomic; and (c) a complete *infinite* test suite is extracted from the specification (see e.g. [30], [31], [32], [26]). In particular, a test derivation algorithm is defined in such a way that, for all implementation behaviors that must be tested before granting the correctness, a suitable test for checking this behavior is added by the algorithm after executing it some finite time. 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 the (c) 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).

The rest of the paper is structured as follows. In Section II we introduce our model. In Section III we give our timed conformance relations and provide several examples to show the differences among them. In Section IV we show how tests are defined and applied to implementations. In Section V we present an algorithm to derive sound and complete test suites with respect to the implementation relations that we presented in Section III. Finally, in Section VI we present our conclusions and some directions for future work.

II. A TIMED EXTENSION OF THE EFSM MODEL

In this section we introduce a new notion of *timed extended finite state machine*. As we have indicated in the introduction of this paper, we will add new features so that the timed behavior of a system can be properly specified. On the one hand, we consider that output actions take time to be executed. These time values will not only depend on the corresponding action to be performed and the current state of the machine. Actually, we will also consider that these time values take into account the current value of the variables. For instance, by using this approach, we can simulate that the speed with which a task is performed depends on the available resources. These resources, more properly, the amount of each resource owned by the corresponding process, can be encoded as a tuple of variables in our model. This will be done in the forthcoming Definition 2 by considering a domain $\mathcal{D}$, defined

as the Cartesian product of different data domains, to represent values. In this line, [32] presents such an approach to deal with resources since the process algebra PAMR [32], useful to specify systems where resources play a fundamental role, can be encoded into a notion of timed EFSM. On the other hand, we will also consider that the machine can evolve by raising *timeouts*. Intuitively, if after a given amount of time and depending on the current state, we do not receive any input action then the machine will change its current state. During the rest of the paper we will use the following notation.

Definition 1: A tuple of elements $(a_1, a_2 \ldots, a_n)$ will be denoted by $\bar{a}$. We say that $\hat{a} = [a_1, a_2)$ is an interval if $a_1, a_2 \in \mathbb{N}$ and $a_1 \leq a_2$. A tuple of intervals $(\hat{p_1}, \ldots, \hat{p_n})$ will be denoted by $\breve{p}$. Let $\bar{t} = (t_1, \ldots, t_n)$, $\bar{t}' = (t'_1, \ldots, t'_n)$, and $\breve{q} = (\hat{q_1}, \ldots, \hat{q_n})$. We write

- $\bar{t} \in \breve{q}$ if for all $1 \leq j \leq n$ we have $t_j \in \hat{q_j}$;
- $\bar{t} = \bar{t}'$ if for all $1 \leq j \leq n$ we have $t_j = t'_j$;
- $\bar{t} \leq \bar{t}'$ if for all $1 \leq j \leq n$ we have $t_j \leq t'_j$.

In addition, we introduce the following notation

- $\sum \bar{t}$ denotes the sum of the elements belonging to the tuple $\bar{t}$, that is, $\sum_{j=1}^{n} t_j$;
- $\pi_i(\bar{t})$, for all $1 \leq i \leq n$, denotes the value t_i.

□

Definition 2: Let $D_1, \ldots, D_m$ be sets of values and let us consider $\mathcal{D} = D_1 \times D_2 \times \cdots \times D_m$. We have that $\mathcal{D}$ is the domain where variables take values. We denote by Time the domain to define time values. A *Timed Extended Finite State Machine*, in the following TEFSM, is a tuple $M = (S, I, O, TO, Tr, s_{in}, \bar{y})$ where S is a finite set of *states*, I is the set of *input action*s, O is the set of *output actions*, $TO : S \longrightarrow S \times (\texttt{Time} \cup \infty)$ is the *timeout function*, Tr is the set of *transitions*, s_{in} is the *initial state*, and $\bar{y} \in \mathcal{D}$ is the tuple of initial values of the *variables*.

A *transition* is a tuple (s, s', i, o, Q, Z, C) where $s, s' \in S$ are the initial and final state of the transition, $i \in I$ and $o \in O$ are the input and output action associated with the transition, $Q : \mathcal{D} \longrightarrow \texttt{Bool}$ is a predicate on the set of variables, $Z : \mathcal{D} \longrightarrow \mathcal{D}$ is a transformation over the variables, and $C : \mathcal{D} \longrightarrow \texttt{Time}$ is the time that the transition needs to be completed.

A *configuration* in M is a pair $(s, \bar{x})$ where $s \in S$ is the current state and $\bar{x} \in \mathcal{D}$ is the tuple containing the current values of the variables.

We say that M is *input-enabled* if for all configuration $(s, \bar{x})$ and input $i \in I$, there exist s', o, Q, Z, C such that $Q(\bar{x})$ holds and $(s, s', i, o, Q, Z, C) \in Tr$. □

Given a configuration $(s, \bar{x})$, a transition (s, s', i, o, Q, Z, C) denotes that if the input i is received and $Q(\bar{x})$ holds then the output o will be produced after $C(\bar{x})$ units of time, and the configuration will be $(s', Z(\bar{x}))$. In this paper we consider that time is discrete. In particular, we will sometimes enumerate the elements of Time simply as $0, 1, 2$ and so on. Finally, let us remark that, even though we have variables, we do not consider input or output parameters in our transitions.

For each state $s \in S$, the application of the timeout function $TO(s)$ returns a pair (s', t) indicating the time that the machine can remain at the state s waiting for an input action, and the state to which the machine evolves if no input is received on time. We assume that $TO(s) = (s', t)$ implies $s \neq s'$, that is, timeouts always produce a change of the state. We indicate the absence of a timeout in a given state by setting the corresponding time value to ∞.

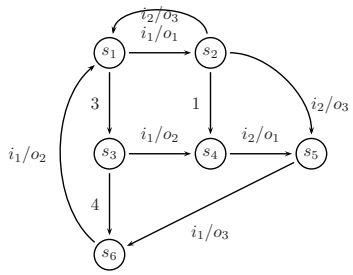
Let us comment on our notion of *input enabling*. In untimed models based on EFSMs, a machine is considered to be input enabled if it can accept an input at any point of time. In the case of timed systems, this notion has to be conveniently adapted. We consider that the machine has to accept any input once it is placed in a state, that is, it is not performing a transition (timeout transitions are instantaneous). Informally, we can see this notion of input enabling as a user interacting with a system through a console. When the prompt symbol appears, the user can write certain commands and the system must react in some way (performing a computation, giving an error message, etc). When the system is occupied performing an operation, the prompt symbol disappears and keyword interactions are ignored.

Finally, let us remark that, in this paper, we are not considering a framework where communicating machines can be defined. Thus, we assume that inputs are provided by the environment and outputs are sent to the environment. However, it is not difficult to extend the current framework to deal with communications between different machines by adapting the framework presented in [32] to the timed model presented in this paper.

Definition 3: Let $M = (S, I, O, TO, Tr, s_{in}, \bar{y})$ be a TEFSM and $c_0 = (s_0, \bar{x}_0)$ be a configuration of M. A tuple $(s_0, s, i/o, \hat{t}, t_o, \bar{v})$ is a *step* of M for the configuration c_0 if there exist $k \geq 0$ states $s_1, \ldots, s_k \in S$ such that for all $1 \leq j \leq k$ we have $TO(s_{j-1}) = (s_j, t_j)$ and there exists a transition $(s_k, s, i, o, Q, Z, C) \in T_r$ such that $Z(\bar{x}_0) = \bar{v}$, $\hat{t} = \left[\sum_{j=1}^{k} t_j, \sum_{j=1}^{k} t_j + \pi_2(TO(s_k))\right)$, $t_o = C(\bar{x}_0)$, and $Q(\bar{x}_0)$ holds.

We say that $(\hat{t}_1/i_1/t_{o1}/o_1, \ldots, \hat{t}_r/i_r/t_{or}/o_r)$ is a *timed evolution* of M if there exist r steps of M $(s_{in}, s_1, i_1/o_1, \hat{t}_1, t_{o1}, \bar{y}_1), \ldots, (s_{r-1}, s_r, i_r/o_r, \hat{t}_r, t_{or}, \bar{y}_r)$ for the configurations $(s_{in}, \bar{y}), \ldots, (s_{r-1}, \bar{y}_{r-1})$, respectively. We denote by $\texttt{TEvol}(M)$ the set of timed evolutions of M. In addition, we say that $(\hat{t}_1/i_1/o_1, \ldots, \hat{t}_r/i_r/o_r)$ is a *functional evolution* of M. We denote by $\texttt{FEvol}(M)$ the set of functional evolutions of M. □

Intuitively, a step is a transition preceded by zero or more timeouts. The interval $\hat{t}$ indicates the time values where the input action could be received. A functional evolution is a sequence of inputs/outputs corresponding to the transitions of a chain of steps. The first of these steps begins with the initial configuration of the machine. These steps include the time interval, indicated by the different intervals $\hat{t}_j$, when each input i_j could be accepted. As we will explain later when we introduce our functional implementation relation, see forthcoming Definition 5, functional evolutions need to include time information. Specifically, they must contain information related to the triggered timeouts. This is due to the fact that timeouts influence the different input/output sequences that TEFSMs can perform. This information is encoded into the intervals where input actions can be performed.



$$
\begin{array}{lcl}
t_{12} & = & (s_1, s_2, i_1, o_1, Q_1, Z_1, C_1) \\
t_{34} & = & (s_3, s_4, i_1, o_2, Q_2, Z_2, C_2) \\
t_{25} & = & (s_2, s_5, i_2, o_3, Q_3, Z_3, C_3) \\
t_{45} & = & (s_4, s_5, i_2, o_1, Q_4, Z_4, C_4) \\
t_{56} & = & (s_5, s_6, i_1, o_3, Q_5, Z_5, C_5) \\
t_{61} & = & (s_6, s_1, i_1, o_2, Q_6, Z_6, C_6) \\
t_{21} & = & (s_2, s_1, i_2, o_3, Q_7, Z_7, C_7)
\end{array}
$$

$$TO(s_1) = (s_3, 3),\ TO(s_3) = (s_6, 4),\ TO(s_2) = (s_4, 1)$$

$$Z_i(\bar{x}) = \bar{x} + \begin{cases} (1,0,0,-1) & \text{if } i \in \{1,3,5\} \\ (0,1,-1,0) & \text{if } i \in \{2,4,6,7\} \end{cases}$$

$$Q_i(\bar{x}) \equiv Z_i(\bar{x}) \geq \bar{0} \wedge \begin{cases} x_i > 0 & \text{if } i \in \{1,2,3,4\} \\ x_1 > 0 & \text{if } i \in \{5,6,7\} \end{cases}$$

$$C_i(\bar{x}) = \begin{cases} x_i + 2 & \text{if } i \in \{1,2,3,4\} \wedge x_i \neq 0 \\ x_1 & \text{if } i \in \{5,6,7\} \wedge x_1 \neq 0 \\ 3 & \text{otherwise} \end{cases}$$

Fig. 1. Example of TEFSM.

In addition to the previous information, timed evolutions present the time consumed to execute each output after receiving each input in each step of the evolution.

Example 1: Let us consider the TEFSM depicted in Figure 1. We suppose that the variables of the TEFSM are given by a tuple $\bar{x} \in \mathbb{R}^4_+$ and we denote by x_i the *i-th* component of $\bar{x}$. Let us assume that initially $\bar{x} = (1, 2, 2, 1)$. Next, we give some of the *steps* that the machine can generate. For example, $(s_1, s_2, i_1/o_1, [0, 3), 3, (2, 2, 2, 0))$, represents the transition t_{12} when no timeouts precede it. The input i_1 can be accepted before 3 time units pass (this is indicated by the interval $[0, 3)$). In addition, o_1 takes $C_1((1, 2, 2, 1)) = 3$ time units to be performed and the new values of $\bar{x}$ are $Z_1((1, 2, 2, 1)) = (2, 2, 2, 0)$. The second one, $(s_1, s_4, i_1/o_2, [3, 7), 4, (1, 3, 1, 1))$ is built from the timeout transition associated to the state s_1 and the transition t_{34}. The step represents that if after 3 time units no input is received then the timeout transition associated with that state will be triggered and the state will change to s_3. After this, the machine can accept the input i_1 before 4 time units pass, that is, the timeout assigned to the state s_3. Therefore, during the time interval $[3, 7)$ if the machine receives an input i_1 it will emit an output o_2 and the state will change to s_4. Similarly, we can obtain the step $(s_1, s_1, i_1/o_2, [7, \infty), 1, (1, 3, 1, 1))$, using the timeout transitions corresponding to s_1 and s_3 and the transition t_{61}. All the steps presented in this example correspond to the initial configuration $(s_1, (1, 2, 2, 1))$.

Now, we present an example of a timed evolution built from two steps and assuming that s_1 is the initial state: $([7, \infty)/i_1/1/o_2, [3, 7)/i_1/5/o_2)$. The configuration that has been considered for the first step is again $(s_1, (1, 2, 2, 1))$. The configuration that corresponds to the second step is the one obtained after the first step has been performed, that is, $(s_1, (1, 3, 1, 1))$. □

In the following definition we introduce the concept of *instanced evolution*. Intuitively, instanced evolutions are constructed from evolutions by instantiating to a concrete value each timeout, given by an interval, of the evolution.

Definition 4: Let $M = (S, I, O, TO, Tr, s_{in}, \bar{y})$ be a TEFSM and $e = (\hat{t}_1/i_1/t_{o1}/o_1, \ldots, \hat{t}_r/i_r/t_{or}/o_r)$ be a *timed evolution* of M. We say that the tuple $(t_1/i_1/t_{o1}/o_1, \ldots, t_r/i_r/t_{or}/o_r)$ is an *instanced timed evolution of* e if for all $1 \leq j \leq r$ we have $t_j \in \hat{t}_j$. In addition, we say that the tuple $(t_1/i_1/o_1, \ldots, t_r/i_r/o_r)$ is an *instanced functional evolution* of e.

We denote by $\texttt{InsTEvol}(M)$ the set of instanced timed evolutions of M and we denote by $\texttt{InsFEvol}(M)$ the set of instanced functional evolutions of M. □

We will sometimes refer to the tuple $(t_1/i_1/t_{o1}/o_1, \ldots, t_r/i_r/t_{or}/o_r) \in \texttt{InsTEvol}(M)$ as $(\bar{t}, \sigma, \bar{t}_o)$, where $t = (t_1, \ldots, t_r)$, $\sigma = (i_1/o_1, \ldots, i_r/o_r)$, and $t_o = (t_{o1}, \ldots, t_{or})$. Similarly, we will also refer to instanced functional evolutions as $(\bar{t}, \sigma)$.

Example 2: If we consider the timed evolution $([7, \infty)/i_1/1/o_2, [3, 7)/i_1/5/o_2)$ showed in the previous example, we have that the tuples $(8, /i_1/1/o_2, 5/i_1/5/o_2)$ and $(12, /i_1/1/o_2, 3/i_1/5/o_2)$ are two of its instanced timed evolutions. □

III. (TIMED) IMPLEMENTATION RELATIONS

In this section we introduce our implementation relations. Intuitively, an implementation is related to a specification if and only if the implementation is *correct* with respect to the specification. However, in our context, the notion of correctness has several possible definitions. In fact, in the case of timed systems, what is a good implementation is even less precise than in untimed systems. For example, one may consider that a system I is a good implementation of a specification S if I takes the same time to perform its tasks as the specification 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.

Even though there are different perspectives to consider what a good implementation is, there is an agreement on correctness if we consider only *functional behavior*, that is, abstracting the time that actions take to be performed. Regarding the performance of usual inputs and outputs, an implementation should not *invent* behaviors when inputs specified in the specification are applied to it. In formal terms, this means that the *relevant* evolutions of the implementation must be contained in those corresponding to the specification. Thus, all our implementation relations follow the same pattern: An implementation I *conforms* to a specification 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 for the specification.

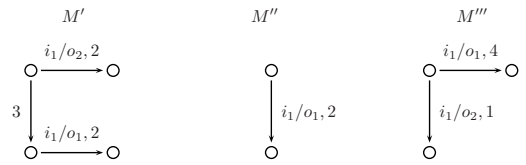


Fig. 2. Examples of functional conformance.

Additionally, we require that the implementation always complies in a certain manner with the timeouts established by the specification. This is due to the fact that timeouts have to be part of this *untimed* relation because they can influence the functional behavior of a system, that is, they influence the actions that a TEFSM can perform at a given point of time. This is illustrated in the following example.

Example 3: Let us consider the schematic machines depicted in Figure 2. These diagrams represent simplified TEFSMs where data is not used. We consider the following notation: A transition labelled by '$i/o, t$' denotes that the execution of the output action o takes time t to be performed after the input i is received; a transition with a label t indicates that a timeout will be applied at time t, that is, if after t time units no expected input is received then the timeout is executed.

We will have that M' is not functionally conforming to M''. The sequence i_1/o_2, that can be performed by M', cannot be performed by M''. If we consider the conformance of M'' with respect to M' and we only check the possible sequences of inputs/outputs, M'' would conform to M' due to the fact that the unique sequence that can be performed by M'' is i_1/o_1. However, this sequence is allowed by M' only in the case that the input has been received after three time units. So, under our conformance framework, M'' does not conform to M'. We can say the same regarding the conformance of M''' with respect to M'. On the contrary, this is not the case when considering the conformance of M' with respect to M'''. The sequences performed by M' are accepted by M''' at any time. So, M' functionally conforms to M'''. □

In our framework, specifications will be modelled by using timed extended finite state machines. Let us note that we consider black-box testing, that is, a system will be tested with respect to a specification without having information about the internal structure of the system. The only knowledge about the IUT (implementation under test) must be inferred from the outputs observed when tests interact with the system. As usual, we also consider that the IUT is described by using a TEFSM having the same sets of inputs and outputs as the specification. We also assume that implementations are input enabled. Let us note that we do not restrict the machines to be deterministic. Thus, both implementations and specifications may present non-deterministic behavior. In addition to increasing the expressive power of the framework, the presence of non-determinism also allows us to define more implementation relations. These facts augment the usability of our framework, being an important advantage with respect to previous work [17].

First, we introduce the implementation relation $\mathtt{conf}_f$, where only functional aspects of the system (i.e., which outputs are allowed/forbidden) are considered while the performance of the system (i.e., how fast are actions executed) is ignored. Let us note that the time spent by a system waiting for the environment to react has the capability of affecting the set of available outputs of the system. This is so because this time may trigger a change of the state by raising one or more timeouts. So, a relation focusing on functional aspects must explicitly take into account the maximal time the system may stay in each state. This time is given by the *timeout* of the state. This is the reason why we have to partially take into account time aspects in the definition of the following *functional* implementation relation: As commented before, raising timeouts may influence the actions available at a given point.

Definition 5: Let S and I be two TEFSMs. We say that I *functionally conforms* to S, denoted by $I\ \mathtt{conf}_f\ S$, if for all $e \in \mathtt{FEvol}(S)$, with $e = (\hat{t}_1/i_1/o_1, \ldots, \hat{t}_r/i_r/o_r)$ and $r \geq 1$, we have that for all $t_1 \in \hat{t}_1, \ldots, t_r \in \hat{t}_r$ and o'_r

$$e' = (t_1/i_1/o_1, \ldots, t_r/i_r/o'_r) \in \mathtt{InsFEvol}(I)$$
$$\Downarrow$$
$$e' \in \mathtt{InsFEvol}(S)$$

□

Let us note that if the specification has also the property of input-enabled then we may remove the condition "for each $e \in \mathtt{FEvol}(S)$, with $e = (\hat{t}_{t1}/i_1/o_1, \ldots, \hat{t}_{tr}/i_r/o_r)$ and $r \geq 1$".

Example 4: Let us consider the schematic machines depicted in Figure 3. Let us note that the behavior of M_1 and M_2 is exactly the same regardless of the timeout presented in M_1. All transitions available for M_1 after taking a timeout are also available in M_2 from its first state. Thus, we have $M_1\ \mathtt{conf}_f\ M_2$. Actually, we also have $M_2\ \mathtt{conf}_f\ M_1$. For similar reasons, we have $M_1\ \mathtt{conf}_f\ M_9$, $M_9\ \mathtt{conf}_f\ M_1$, $M_2\ \mathtt{conf}_f\ M_9$, and $M_9\ \mathtt{conf}_f\ M_2$.

Next, we show how the availability of outputs affects the functional relation $\mathtt{conf}_f$. We have $M_5\ \mathtt{c\not{o}nf}_f\ M_{11}$. Let us note that if i_2 is offered in M_{11} after executing i_1/o_1 then only o_1 can be produced. However, M_5 can produce this output as well as o_2, which is forbidden by M_{11}. So, we do not have $M_5\ \mathtt{conf}_f\ M_{11}$. If we consider M_6 instead of M_5 then the same considerations apply: We have $M_6\ \mathtt{c\not{o}nf}_f\ M_{11}$. Moreover, we have $M_9\ \mathtt{c\not{o}nf}_f\ M_{10}$ and $M_{10}\ \mathtt{c\not{o}nf}_f\ M_9$. Let us note that after 1 time unit passes and the timeout is raised, if i_1 is offered then M_9 must answer o_1 while o_2 is forbidden. However, it is the other way around for M_{10}. □

In the following definition we introduce our *timed* implementation relations. We will distinguish two classes of conformance relations: *Weak* and *strong*. The family of weak conformance relations demands conditions over the total time associated to instanced timed evolutions of the implementation with respect to the corresponding instanced timed evolutions of the specification. In contrast, strong conformance relations establish requests over the time values corresponding to the performance of each transition separately. For each of these

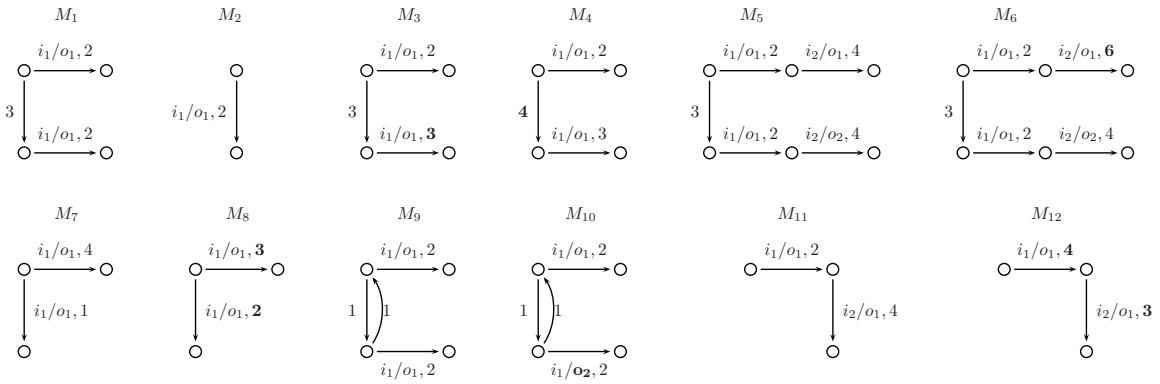


Fig. 3. Examples of schematic machines representing TEFSMs.

approaches we define five relations. Informally, the $\texttt{conf}_a^s$ and $\texttt{conf}_a^w$ relations (conforms *always*) hold if for all instanced timed evolution e of the implementation we have that if its associated instanced functional evolution is an instanced functional evolution of the specification then e is also an instanced timed evolution of the specification. In the $\texttt{conf}_w^s$ and $\texttt{conf}_w^w$ relations (conformance in the *worst* case) the implementation is forced, for each instanced timed evolution fulfilling the previous conditions, to be faster than the slowest instance of the same evolution in the specification. The $\texttt{conf}_b^s$ and $\texttt{conf}_b^w$ relations (conforms in the *best* case) are similar but taking into account only the fastest instance of the specification. Finally, the relations $\texttt{conf}_{sw}^s$ and $\texttt{conf}_{sw}^w$ (*sometimes worst*), and $\texttt{conf}_{sb}^s$ and $\texttt{conf}_{sb}^w$ (*sometimes best*), are similar to the previous relations, but in each case only *one* instance of each timed evolution of the implementation is required to be as fast as the worst/best instance in the specification.

Definition 6: Let $\bar{t}_o = (t_{o1} \dots t_{or}) \in \texttt{Time}^r$. Given an instanced functional evolution $insfevol = (t_1/i_1/o_1, \dots, t_r/i_r/o_r) \in (\texttt{Time} \times I \times O)^r$, we denote by $insfevol \nabla \bar{t}_o$ the instanced timed evolution $(t_1/i_1/t_{o1}/o_1, \dots, t_r/i_r/t_{or}/o_r)$. Let S and I be two TEFSMs. The *timed conformance relations* are defined as follows:

- *(strong always)* $I\ \texttt{conf}_a^s\ S$ iff $I\ \texttt{conf}_f\ S$ and for all $insfevol \in \texttt{InsFEvol}(I) \cap \texttt{InsFEvol}(S)$ we have that $\forall\ \bar{t}_i$

$$insfevol \nabla \bar{t}_i \in \texttt{InsTEvol}(I)$$
$$\Downarrow$$
$$insfevol \nabla \bar{t}_i \in \texttt{InsTEvol}(S)$$

- *(strong best)* $I\ \texttt{conf}_b^s\ S$ iff $I\ \texttt{conf}_f\ S$ and for all $insfevol \in \texttt{InsFEvol}(I) \cap \texttt{InsFEvol}(S)$ we have that $\forall\ \bar{t}_i$

$$insfevol \nabla \bar{t}_i \in \texttt{InsTEvol}(I)$$
$$\Downarrow$$
$$\forall\ \bar{t}_s : (insfevol \nabla \bar{t}_s \in \texttt{InsTEvol}(S) \Longrightarrow \bar{t}_i \leq \bar{t}_s)$$

- *(strong worst)* $I\ \texttt{conf}_w^s\ S$ iff $I\ \texttt{conf}_f\ S$ and for all $insfevol \in \texttt{InsFEvol}(I) \cap \texttt{InsFEvol}(S)$ we have that $\forall\ \bar{t}_i$

$$insfevol \nabla \bar{t}_i \in \texttt{InsTEvol}(I)$$
$$\Downarrow$$
$$\exists\ \bar{t}_s : (insfevol \nabla \bar{t}_s \in \texttt{InsTEvol}(S)\ \wedge\ \bar{t}_i \leq \bar{t}_s)$$

- *(strong sometimes best)* $I\ \texttt{conf}_{sb}^s\ S$ iff $I\ \texttt{conf}_f\ S$ and for all $insfevol \in \texttt{InsFEvol}(I) \cap \texttt{InsFEvol}(S)$ we have that $\exists\ \bar{t}_i$ such that

$$insfevol \nabla \bar{t}_i \in \texttt{InsTEvol}(I)$$
$$\wedge$$
$$\forall\ \bar{t}_s : (insfevol \nabla \bar{t}_s \in \texttt{InsTEvol}(S) \Longrightarrow \bar{t}_i \leq \bar{t}_s)$$

- *(strong sometimes worst)* $I\ \texttt{conf}_{sw}^s\ S$ iff $I\ \texttt{conf}_f\ S$ and for all $insfevol \in \texttt{InsFEvol}(I) \cap \texttt{InsFEvol}(S)$ we have that $\exists\ \bar{t}_i, \bar{t}_s$ such that

$$\left(\begin{array}{c} insfevol \nabla \bar{t}_i \in \texttt{InsTEvol}(I) \\ \wedge \\ insfevol \nabla \bar{t}_s \in \texttt{InsTEvol}(S) \\ \wedge \\ \bar{t}_i \leq \bar{t}_s \end{array} \right)$$

- *(weak always)* $I\ \texttt{conf}_a^w\ S$ iff $I\ \texttt{conf}_f\ S$ and for all $insfevol \in \texttt{InsFEvol}(I) \cap \texttt{InsFEvol}(S)$ we have that $\forall\ \bar{t}_i$

$$insfevol \nabla \bar{t}_i \in \texttt{InsTEvol}(I)$$
$$\Downarrow$$
$$\exists\ \bar{t}_s : \left(insfevol \nabla \bar{t}_s \in \texttt{InsTEvol}(S)\ \wedge\ \sum \bar{t}_i = \sum \bar{t}_s \right)$$

- *(weak best)* $I\ \texttt{conf}_b^w\ S$ iff $I\ \texttt{conf}_f\ S$ and for all $insfevol \in \texttt{InsFEvol}(I) \cap \texttt{InsFEvol}(S)$ we have that $\forall\ \bar{t}_i$

$$insfevol \nabla \bar{t}_i \in \texttt{InsTEvol}(I)$$
$$\Downarrow$$
$$\forall\ \bar{t}_s : \left(insfevol \nabla \bar{t}_s \in \texttt{InsTEvol}(S) \Longrightarrow \sum \bar{t}_i \leq \sum \bar{t}_s \right)$$

- *(weak worst)* $I\ \mathtt{conf}^{w}_{w}\ S$ iff $I\ \mathtt{conf}_{f}\ S$ and for all $insfevol \in \mathtt{InsFEvol}(I) \cap \mathtt{InsFEvol}(S)$ we have that $\forall\ \bar{t}_i$

$$insfevol \nabla \bar{t}_i \in \mathtt{InsTEvol}(I)$$

$$\Downarrow$$

$$\exists\ \bar{t}_s : \left(insfevol \nabla \bar{t}_s \in \mathtt{InsTEvol}(S)\ \wedge\ \sum \bar{t}_i \leq \sum \bar{t}_s \right)$$

- *(weak sometimes best)* $I\,\mathtt{conf}^{w}_{sb}\,S$ iff $I\,\mathtt{conf}_{f}\,S$ and for all $insfevol \in \mathtt{InsFEvol}(I) \cap \mathtt{InsFEvol}(S)$ we have that $\exists\ \bar{t}_i$ such that

$$insfevol \nabla \bar{t}_i \in \mathtt{InsTEvol}(I)$$

$$\wedge$$

$$\forall\ \bar{t}_s : \left(insfevol \nabla \bar{t}_s \in \mathtt{InsTEvol}(S) \Longrightarrow \sum \bar{t}_i \leq \sum \bar{t}_s \right)$$

- *(weak sometimes worst)* $I\ \mathtt{conf}^{w}_{sw}\ S$ iff $I\ \mathtt{conf}_{f}\ S$ and for all $insfevol \in \mathtt{InsFEvol}(I) \cap \mathtt{InsFEvol}(S)$ we have that $\exists\ \bar{t}_i$, $\bar{t}_s$ such that

$$\left(\begin{array}{c} insfevol \nabla \bar{t}_i \in \mathtt{InsTEvol}(I) \\ \wedge \\ insfevol \nabla \bar{t}_s \in \mathtt{InsTEvol}(S) \\ \wedge \\ \sum \bar{t}_i \leq \sum \bar{t}_s \end{array} \right)$$

□

There are several reasons which can produce that an implementation does not conform to a specification. We will give several examples where different systems, depicted in Figure 3, are considered. For the sake of simplicity, we will use some additional conformance binary operators. In the next examples, $I\ \mathtt{conf}_{*}\ S$ denotes that *all* implementation relations given in Definition 6 hold between I and S. If *none* of these relations holds then we denote it by $I\ \mathtt{co\not{n}f}_{*}\ S$. Besides, $I\ \mathtt{conf}_{\Box}\ S$ denotes that all relations, except $\mathtt{conf}^{s}_{a}$ and $\mathtt{conf}^{w}_{a}$, hold.

Example 5: If we consider the machines M_3 and M_2 we have $M_3\ \mathtt{co\not{n}f}_{*}\ M_2$. This is due to the existence of *different time values to perform output actions*. Let us note that M_3 may take 3 time units to perform the output o_1 if it receives the input i_1 after 3 time units, $(3/i_1/3/o_1)$, while M_2 only needs 2 time units, $(3/i_1/2/o_1)$. Since M_3 is, in *any* case, slower than M_2 for these sequences of inputs/outputs, no conformance relation where M_3 is the IUT and M_2 is the specification holds. On the contrary, we have $M_2\ \mathtt{conf}_{\Box}\ M_3$: Despite the fact that M_2 does not take the same time values as M_3 for each sequence, its time is always smaller than (in the case that the input is received after the timeout is triggered) or equal to (in the case that the input is received before the timeout is triggered) the time values corresponding to M_3. □

As we have seen, reducing the time consumed by actions can produce that a certain TEFSM conforms to another one with respect to some of the available implementation relations. In spite of the fact that requirements on timeouts are *strict*, sometimes having different timeouts can also produce the same effect.

Example 6: Let us consider M_3 and M_4. In this case the non timed conformance is produced by the fact that the machines have *different timeouts*. Most input/output sequences in M_3 and M_4 take the same time values to be performed. However, there is an exception: The sequence including the timeout 3. In M_3 we have $(3/i_1/3/o_1)$ but in M_4 we have $(3/i_1/2/o_1)$. This is so because after 3 time units pass the state does not change yet in M_4. Hence, we have $M_4\ \mathtt{conf}_{\Box}\ M_3$ but $M_3\ \mathtt{co\not{n}f}_{*}\ M_4$. □

Let us consider a case where IUTs and specifications can spend different time values in executing pairs of input/outputs included in evolutions.

Example 7: We consider M_7 and M_8. Since they only perform sequences of length 1, strong relations coincide with their respective weak versions. Next we will refer to strong relations. Both M_7 and M_8 can execute i_1/o_1 by spending an amount of time that the other one cannot take. So, we do not have $M_7\ \mathtt{conf}^{s}_{a}\ M_8$. The worst time values to execute i_1/o_1 in M_7 and M_8 are 4 and 3, respectively, while the best time values are 1 and 2, respectively. The worst time of M_7 is not better than either the worst or the best time of M_8. Thus, we have neither $M_7\ \mathtt{conf}^{s}_{w}\ M_8$ nor $M_7\ \mathtt{conf}^{s}_{b}\ M_8$. However, the best time of M_7 is better than both the worst and the best time values of M_8. So, both $M_7\ \mathtt{conf}^{s}_{sw}\ M_8$ and $M_7\ \mathtt{conf}^{s}_{sb}\ M_8$ hold. Moreover, the worst time of M_8 is better than the worst one in M_7 but worst than the best of M_7. Hence, $M_8\,\mathtt{conf}^{s}_{w}\,M_7$ holds but $M_8\,\mathtt{conf}^{s}_{b}\,M_7$ does not. Finally, the best time in M_8 is better than the worst of M_7, but not better than its best one. Thus, $M_8\ \mathtt{conf}^{s}_{sw}\ M_7$ holds, but $M_8\ \mathtt{conf}^{s}_{sb}\ M_7$ does not. All these situations are due to the fact that these machines have *different time requirements*. □

Finally, we show how temporal requirements are dealt with by strong and weak relations.

Example 8: Let us consider M_{11} and M_{12}. No strong relation holds between these TEFSMs in any direction. The reason is that M_{11} performs i_1/o_1 faster than M_{12}, but M_{12} performs the next transition, i_2/o_1, faster than M_{11}. The result is that none of these machines is always at least as fast, concerning transitions, as the other one. However, if we consider complete timed evolutions, instead of their steps separately (i.e., weak relations), then some relations hold. Let us note that M_{11} performs both available sequences of inputs/outputs (i_1/o_1 and $i_1/o_1, i_2/o_1$) faster than M_{12}: In M_{11} they spend 2 and 6 time units, respectively, while these time values are 4 and 7, respectively, in M_{12}. So, all *weak* relations, except $\mathtt{conf}^{w}_{a}$, hold: We have $M_{11}\ \mathtt{conf}^{w}_{w}\ M_{12}$, $M_{11}\ \mathtt{conf}^{w}_{b}\ M_{12}$, $M_{11}\ \mathtt{conf}^{w}_{sw}\ M_{12}$, and $M_{11}\ \mathtt{conf}^{w}_{sb}\ M_{12}$. Let us remark that none of these relations holds if we exchange the roles of both machines. □

Theorem 1: The relations given in Definition 6 are related

as follows:

$$\begin{array}{ccccc} & & I\,\mathtt{conf}^w_{sw}\,S & \Leftarrow & I\,\mathtt{conf}^w_{sb}\,S \\ & & \Uparrow & & \Uparrow \\ I\,\mathtt{conf}^w_{a}\,S & \Rightarrow & I\,\mathtt{conf}^w_{w}\,S & \Leftarrow & I\,\mathtt{conf}^w_{b}\,S \\ \Uparrow & & \Uparrow & & \Uparrow \\ I\,\mathtt{conf}^s_{a}\,S & \Rightarrow & I\,\mathtt{conf}^s_{w}\,S & \Leftarrow & I\,\mathtt{conf}^s_{b}\,S \\ & & \Downarrow & & \Downarrow \\ & & I\,\mathtt{conf}^s_{sw}\,S & \Leftarrow & I\,\mathtt{conf}^s_{sb}\,S \end{array}$$

Besides, we have

$$I\,\mathtt{conf}^s_{sw}\,S \Rightarrow I\,\mathtt{conf}^w_{sw}\,S$$

and

$$I\,\mathtt{conf}^s_{sb}\,S \Rightarrow I\,\mathtt{conf}^w_{sb}\,S$$

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 functional conformance is the same in all the definitions. So, we only need to take into account the conditions on time values appearing in the second clause of the corresponding relations. If $I\,\mathtt{conf}^s_a\,S$ then we have that each timed evolution in I fulfilling the conditions given in the definition of $\mathtt{conf}^s_a$ does also appear in S. So, we have $I\,\mathtt{conf}^s_w\,S$. If $I\,\mathtt{conf}^s_b\,S$ then each timed evolution of I fulfilling the conditions given in the definition of $\mathtt{conf}^s_b$ is faster than the fastest instance of the same evolution of S. Therefore, it is also faster than the slowest one of S. So, we conclude $I\,\mathtt{conf}^s_w\,S$.

If $I\,\mathtt{conf}^s_w\,S$ then we know that each instance of a timed 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 $\mathtt{conf}^s_{sw}$. So, we conclude $I\,\mathtt{conf}^s_{sw}\,S$. The same reasoning can be also used to prove that $I\,\mathtt{conf}^s_b\,S$ implies $I\,\mathtt{conf}^s_{sb}\,S$.

If $I\,\mathtt{conf}^s_{sb}\,S$ 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, we conclude $I\,\mathtt{conf}^s_{sw}\,S$.

Regarding the implications relating weak conformance relations we can argue in the same way.

Finally, it only remains to study the implications between strong and weak conformance relations. The reasoning is equivalent. The differences between them consist in the way temporal conditions must be fulfilled. While strong relations demand conditions over the time value of each transition that composes an evolution, weak relations do it over the total time of the evolution. In fact, it is straightforward to check that if the different conditions fulfill for each of the transitions then they fulfill for the corresponding evolution containing all those transitions. Hence, we have that if $I\,\mathtt{conf}^s_a\,S$ then $I\,\mathtt{conf}^w_a\,S$, if $I\,\mathtt{conf}^w_w\,S$ then $I\,\mathtt{conf}^s_w\,S$ and $I\,\mathtt{conf}^s_b\,S$ implies $I\,\mathtt{conf}^w_b\,S$. For the same reason, we have that $I\,\mathtt{conf}^s_{sw}\,S \Rightarrow I\,\mathtt{conf}^w_{sw}\,S$ and $I\,\mathtt{conf}^s_{sb}\,S \Rightarrow I\,\mathtt{conf}^w_{sb}\,S$. □

Let us remark that the implications inferred in the previous result are, obviously, transitive. For instance, we also have $I\,\mathtt{conf}^w_a\,S \Rightarrow I\,\mathtt{conf}^w_{sw}\,S$. Let us also note that if specifications are deterministic then, on the one hand, the relations $\mathtt{conf}^w_b$ and $\mathtt{conf}^w_w$ coincide while, on the other hand, $\mathtt{conf}^s_b$ and $\mathtt{conf}^s_w$ also coincide. However these relations would be still different from the $\mathtt{conf}^w_a$ and $\mathtt{conf}^s_a$ relations. Similarly, if this property holds in implementations then all relations concerning the best instanced timed evolutions of the implementation (*sometimes* relations) coincide with the corresponding relation where all the instanced timed evolutions are taken into account.

Lemma 1: Let $I = (S_I, I_I, O_I, TO_I, Tr_I, s_{in_I}, \bar{y}_I)$ and $S = (S_S, I_S, O_S, TO_S, Tr_S, s_{in_S}, \bar{y}_S)$ be two TEFSMs. If there do not exist different transitions (s, s', i, o, Q, Z, C) and $(s, s'', i, o, Q', Z', C')$ belonging to Tr_S then

$$\begin{array}{ll} I\,\mathtt{conf}^s_b\,S \Leftrightarrow I\,\mathtt{conf}^s_w\,S & I\,\mathtt{conf}^s_{sw}\,S \Leftrightarrow I\,\mathtt{conf}^s_{sb}\,S \\ I\,\mathtt{conf}^w_b\,S \Leftrightarrow I\,\mathtt{conf}^w_w\,S & I\,\mathtt{conf}^w_{sw}\,S \Leftrightarrow I\,\mathtt{conf}^w_{sb}\,S \end{array}$$

If there do not exist different transitions (s, s_1, i, o, Q, Z, C) and $(s, s_2, i, o, Q', Z', C')$ belonging to Tr_I then

$$\begin{array}{ll} I\,\mathtt{conf}^s_b\,S \Leftrightarrow I\,\mathtt{conf}^s_{sb}\,S & I\,\mathtt{conf}^s_w\,S \Leftrightarrow I\,\mathtt{conf}^s_{sw}\,S \\ I\,\mathtt{conf}^w_b\,S \Leftrightarrow I\,\mathtt{conf}^w_{sb}\,S & I\,\mathtt{conf}^w_w\,S \Leftrightarrow I\,\mathtt{conf}^w_{sw}\,S \end{array}$$

□

The hierarchy of relations induced by Theorem 1 allows us to compare implementations in the following way: I_1 is *preferable* to I_2 to implement S if the former meets with S a *stricter* relation.

Definition 7: Let I_1, I_2, and S be TEFSMs and $\mathtt{conf}_x$ and $\mathtt{conf}_y$ be timed conformance relations such that $I_1\,\mathtt{conf}_x\,S$, $I_2\,\mathtt{conf}_y\,S$, $\mathtt{conf}_x \Rightarrow \mathtt{conf}_y$, and $\mathtt{conf}_y \not\Rightarrow \mathtt{conf}_x$. We say that I_1 is *preferred* to I_2 to implement S and we denote it by $I_1 >_S I_2$. □

IV. Definition and Application of Tests

In our setting, tests represent sequences of inputs applied to an IUT. Once an output is received, it must be checked whether it belongs to the set of expected ones or not. In addition to check the functional behavior of the IUT, tests have also to detect whether wrong timed behaviors appear. Thus, tests have to include capabilities to deal with both ways of specifying time. On the one hand, we will include *time stamps*. The time values recorded from the IUT while applying the test will be compared with the ones expected by the specification. Each time stamp will contain a set of *time sequences* corresponding to the time values that the specification establishes for each transition of a timed evolution. Since we do not restrict non-deterministic behavior, we will have as many time sequences as possible timed evolutions can exist for a given input/output sequence. Moreover, depending on the number of inputs applied so far, we will have different lengths for the associated time sequences appearing in the time stamps of the test. For example, if we reach a pass state of the test after applying 3 inputs, and receiving 3 outputs, then the state will include as time stamp a sequence containing 3 time values. In addition, tests will include *delays* before offering input actions. The purpose of delays is to induce timeouts in the

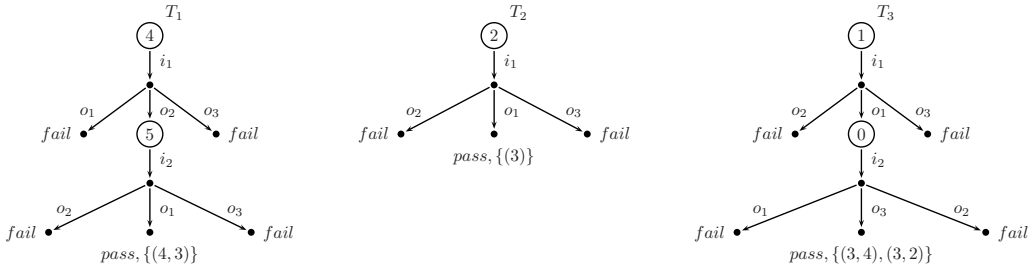


Fig. 4. Examples of Tests.

tested machine. For example, let us consider to test whether a machine conforms to M_{10}, as defined in Figure 3. We can define a test inducing a delay of 0 time units in the tested machine and checking that after applying i_1 we receive o_1. Another test might apply a delay of 1 time unit; afterwards, we should observe o_2 after applying the same input i_1. Thus, we may indirectly check whether the timeouts imposed by the specification are reflected in the IUT by offering input actions after a specific delay. Let us note that we cannot observe when the IUT takes a timeout. However, it is still possible to check the IUT behavior after different delays. Finally, let us remark that our tests have a more complex structure than being a sequence of inputs (and its corresponding expected outputs). This situation also appears in other testing frameworks (see, for example, [33] where quasi-reduction and quasi-equivalence are studied in the framework of nondeterministic FSMs).

Definition 8: A *test* is a tuple $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C, W)$ 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 constitute a partition of S, that is, these four sets represent a division of S into non-overlapping parts that cover all 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*. Thus, 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, we have two timed functions. $C : S_P \longrightarrow \bigcup_{j=1}^{\infty} \mathcal{P}(\texttt{Time}^j)$ is a function associating time stamps with pass states and $W : S_I \longrightarrow \texttt{Time}$ is a function associating delays with input states.

We say that a test T is *valid* if the graph induced by T is a tree with root at the initial state s_0. In the rest of the paper we consider only valid tests.

Let $\sigma = i_1/o_1, \ldots, i_r/o_r$. We write $T \stackrel{\sigma}{\Longrightarrow} s^T$ if $s^T \in S_F \cup S_P$ and 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 $(s_{j2}, o_j, s_{(j+1)1}) \in Tr$.

Let T be a test, $\sigma = i_1/o_1, \ldots, i_r/o_r$, s^T be a state of T, and $\overline{t} \in \texttt{Time}^r$. We write $T \stackrel{\sigma}{\Longrightarrow}_{\overline{t}} s^T$ if $T \stackrel{\sigma}{\Longrightarrow} s^T$, $t_1 = W(s_0)$ and for all $1 < j \leq r$ we have $t_j = W(s_{j1})$. □

In Figure 4 we show a graphical representation of some tests. Let us remark that $T \stackrel{\sigma}{\Longrightarrow} s^T$, and its variant $T \stackrel{\sigma}{\Longrightarrow}_{\overline{t}} s^T$, imply that s^T is a terminal state. Next we define the application of a test suite to an implementation. We say that the test suite $\mathcal{T}$ is *passed* if for all test belonging to the suite we have that the terminal states reached by the composition of implementation and test are *pass* states. Besides, we give different timing conditions in a similar way to what we did for implementation relations.

Definition 9: Let I be a TEFSM, T be a valid test, $\sigma = i_1/o_1, \ldots, i_r/o_r$, s^T be a state of T, $\overline{t} = (t_1, \ldots, t_r)$, and $\overline{t}_o = (t_{o1}, \ldots, t_{or})$. We write $I \parallel T \stackrel{\sigma}{\Longrightarrow}_{\overline{t}} s^T$ if $T \stackrel{\sigma}{\Longrightarrow}_{\overline{t}} s^T$ and $(\overline{t}, \sigma) \in \texttt{InsFEvol}(I)$. We write $I \parallel T \stackrel{\sigma}{\Longrightarrow}_{\overline{t},\overline{t}_o} s^T$ if $I \parallel T \stackrel{\sigma}{\Longrightarrow}_{\overline{t}} s^T$ and $(\overline{t}, \sigma, \overline{t}_o) \in \texttt{InsTEvol}(I)$. Let $e = (\overline{t}, \sigma, \overline{t}_o) \in \texttt{InsTEvol}(I)$. We define the set $\texttt{Test}(e, \mathcal{T}) = \{(T, s^T) \mid T \in \mathcal{T} \wedge I \parallel T \stackrel{\sigma}{\Longrightarrow}_{\overline{t},\overline{t}_o} s^T\}$. We say that

- I *passes* the test suite $\mathcal{T}$, denoted by $\texttt{pass}(I, \mathcal{T})$, if for all test $T \in \mathcal{T}$ there do not exist σ, s^T, $\overline{t}$ such that $I \parallel T \stackrel{\sigma}{\Longrightarrow}_{\overline{t}} s^T$ and $s^T \in S_F$.
- I *strongly passes* the test suite $\mathcal{T}$ *for any time* if $\texttt{pass}(I, \mathcal{T})$ and for all $e = (\overline{t}, \sigma, \overline{t}_o) \in \texttt{InsTEvol}(I)$ we have that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$, $\overline{t}_o \in C(s^T)$ holds.
- I *strongly passes* the test suite $\mathcal{T}$ *in the best time* if $\texttt{pass}(I, \mathcal{T})$ and for all $e = (\overline{t}, \sigma, \overline{t}_o) \in \texttt{InsTEvol}(I)$ we have that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$ and for all $\overline{t}_c \in C(s^T)$, $\overline{t}_o \leq \overline{t}_c$ holds.
- I *strongly passes* the test suite $\mathcal{T}$ *in the worst time* if $\texttt{pass}(I, \mathcal{T})$ and for all $e = (\overline{t}, \sigma, \overline{t}_o) \in \texttt{InsTEvol}(I)$ we have that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$ there exists $\overline{t}_c \in C(s^T)$ such that $\overline{t}_o \leq \overline{t}_c$ holds.
- I *strongly passes* the test suite $\mathcal{T}$ *sometimes in best time* if $\texttt{pass}(I, \mathcal{T})$ and there exists $e = (\overline{t}, \sigma, \overline{t}_o) \in \texttt{InsTEvol}(I)$ such that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$ we have that for all $\overline{t}_c \in C(s^T)$, $\overline{t}_o \leq \overline{t}_c$ holds.
- I *strongly passes* the test suite $\mathcal{T}$ *sometimes in worst time* if $\texttt{pass}(I, \mathcal{T})$ and there exists $e = (\overline{t}, \sigma, \overline{t}_o) \in \texttt{InsTEvol}(I)$ such that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$

we have that there exists $\bar{t}_c \in C(s^T)$ such that $\bar{t}_o \leq \bar{t}_c$ holds.

- *I weakly passes* the test suite $\mathcal{T}$ *for any time* if $\texttt{pass}(I, \mathcal{T})$ and for all $e = (\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$ we have that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$, $\sum \bar{t}_o = \sum \bar{t}_c$ holds for some $\bar{t}_c \in C(s^T)$.
- *I weakly passes* the test suite $\mathcal{T}$ *in the best time* if $\texttt{pass}(I, \mathcal{T})$ and for all $e = (\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$ we have that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$ and for all $\bar{t}_c \in C(s^T)$, $\sum \bar{t}_o \leq \sum \bar{t}_c$ holds.
- *I weakly passes* the test suite $\mathcal{T}$ *in the worst time* if $\texttt{pass}(I, \mathcal{T})$ and for all $e = (\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$ we have that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$ there exists $\bar{t}_c \in C(s^T)$ such that $\sum \bar{t}_o \leq \sum \bar{t}_c$ holds.
- *I weakly passes* the test suite $\mathcal{T}$ *sometimes in best time* if $\texttt{pass}(I, \mathcal{T})$ and there exists $e = (\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$ such that for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$ we have that for all $\bar{t}_c \in C(s^T)$, $\sum \bar{t}_o \leq \sum \bar{t}_c$ holds.
- *I weakly passes* the test suite $\mathcal{T}$ *sometimes in worst time* if $\texttt{pass}(I, \mathcal{T})$ and there exists $e = (\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$ such that and for all $(T, s^T) \in \texttt{Test}(e, \mathcal{T})$ we have that there exists $\bar{t}_c \in C(s^T)$ such that $\sum \bar{t}_o \leq \sum \bar{t}_c$ holds.

□

V. Test Derivation

In this section we present an algorithm to derive tests from specifications. As usual, the idea underlying our algorithm consists in traversing the specification in order to get all the possible input/output sequences in an appropriate way. First, we introduce some additional notation. The following functions will be used in the forthcoming derivation algorithm.

Definition 10: Let $M = (S, I, O, TO, Tr, s_{in}, \bar{y})$ be a TEFSM. The function $\texttt{out}(s, i, \bar{x})$ computes the set of outputs associated with those transitions that can be executed from s after receiving the input i, and assuming that the value of the variables is given by $\bar{x}$.

$$\texttt{out}(s, i, \bar{x}) = \left\{ o \;\middle|\; \begin{array}{l} \exists\, s' \in S, Q, Z, C : \\ (s, s', i, o, Q, Z, C) \in T_r \\ \wedge\, Q(\bar{x}) \end{array} \right\}$$

We have that $\texttt{afterTO}(s, t, \bar{x}, \bar{t})$ computes the state that will be reached in M if we start in the configuration $(s, \bar{x})$ and t time units pass without receiving an input.

$$\texttt{afterTO}(s, t, \bar{x}, \bar{t}) = \begin{cases} (s, \bar{x}, \bar{t}) & \text{if } \pi_2(t_0) > t \\ \texttt{afterTO}(\pi_1(t_0), t - \pi_2(t_0), \bar{x}, \bar{t}) & \text{otherwise} \end{cases}$$

where we are considering $t_0 = TO(s)$. Let us remind that $TO(s)$ denotes the timeout associated with the state s, that is, a pair containing the reached state after the performance of the timeout and when the timeout will be triggered.

The function $\texttt{after}(s, i, o, \bar{x}, \bar{t})$ computes all the states that can be reached from a state s after receiving the input i, producing the output o, for a value of the variables $\bar{x}$, and assuming that the time values included in $\bar{t} = (t_1, \ldots, t_n)$ denote the different time values needed to perform previous actions. In addition, this function also returns the new value of the variables and the updated tuple of time values consumed since the system started its performance.

$$\texttt{after}(s, i, o, \bar{x}, \bar{t}) = \left\{ (s', \bar{x}', \bar{t}') \;\middle|\; \begin{array}{l} \exists\, Q, Z, C : \\ (s, s', i, o, Q, Z, C) \in T_r \\ \wedge\, Q(\bar{x}) \;\wedge\; Z(\bar{x}) = \bar{x}' \;\wedge\; C(\bar{x}) = t \\ \wedge\, \bar{t}' = (t_1, \ldots, t_n, t) \end{array} \right\}$$

□

The previously defined functions can be extended in the natural way to deal with sets:

$$\begin{array}{rcl} \texttt{out}(G, i) & = & \bigcup_{(s,\bar{x}) \in G} \texttt{out}(s, i, \bar{x}) \\ \texttt{afterTO}(D, t) & = & \{\texttt{afterTO}(s, t, \bar{x}, \bar{t}) \mid (s, \bar{x}, \bar{t}) \in D\} \\ \texttt{after}(D, i, o) & = & \bigcup_{(s,\bar{x},\bar{t}) \in D} \texttt{after}(s, i, o, \bar{x}, \bar{t}) \end{array}$$

The algorithm to derive tests from a specification is given in Figure 5. This algorithm is non-deterministic and its application generates a single test. By considering all the possible non-deterministic choices in the algorithm we extract a full test suite from the specification. Let us remark that this set will be, in general, infinite. For a given specification M, we denote this test suite by $\texttt{tests}(M)$. Essentially, our algorithm consists in traversing the specification M in all the possible ways. In addition to generate tests that take into account the outputs that can be followed by the application of an input, the tests will also include temporal information regarding the time values associated with the performance of these outputs in the specification.

Next we explain how the algorithm works. A set of *pending situations* D keeps those triples denoting the states that could have been reached in the transversal of the specification, the corresponding value of the variables, and sequences of time values that could appear in a state of the test whose definition (that is, the construction of 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 in the specification 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 specification.

Example 9: Let $M = (S, I, O, TO, Tr, s_{in}, \bar{y})$ be a TEFSM. Let us suppose that we have two transitions $(s, s', i, o, Q_1, Z_1, C_1), (s, s'', i, o, Q_2, Z_2, C_2) \in Tr$. If we want 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 values consumed for each transition so far are given by $\bar{t} = (t_1, \ldots, t_n)$, we have to compute the set $\texttt{after}(\{(s, \bar{x}, \bar{t})\}, i, o)$. As we explained before, the application of this function will return the different situations that can be reached by the specification after receiving the input i and generating the output o, considering that the current configuration of the machine

Input: A specification $M = (S, I, O, TO, Tran, s_{in}, \bar{y})$.
Output: A test case $T = (S', I, O, Tran', s_0, S_I, S_O, S_F, S_P, C, W)$.

Initialization:

- $S' := \{s_0\}, Tran' := S_I := S_O := S_F := S_P := \emptyset$.
- $S_{aux} := \{(\{(s_{in}, \bar{y}, <>)\}, s_0)\}$.

Inductive Cases: Choose one of the following two options until $S_{aux} = \emptyset$.

1) If $(D, s^T) \in S_{aux}$ then perform:
   a) $S_{aux} := S_{aux} - \{(D, s^T)\}$.
   b) $S_P := S_P \cup \{s^T\}$.
   c) $C(s^T) := \{\bar{t} \mid (s, \overline{x}, \bar{t}) \in D\}$.
2) If $S_{aux} = \{(D, s^T)\}$ and $\exists t_d \in \texttt{Time}, i \in I$ such that $\texttt{out}(S_M, i) \neq \emptyset$, with $S_M = \{(s, \bar{x}) \mid (s, \bar{x}, \bar{t}) \in \texttt{afterTO}(D, t_d)\}$, then perform:
   a) Choose $t_d \in \texttt{Time}$ and $i \in I$ fulfilling the previous conditions.
   b) $D := \texttt{afterTO}(D, t_d)$.
   c) $S_M := \{(s, \bar{x}) \mid (s, \bar{x}, \bar{t}) \in D\}$.
   d) $S_{aux} := \emptyset$.
   e) Consider a fresh state $s' \notin S'$ and let $S' := S' \cup \{s'\}$.
   f) $S_I := S_I \cup \{s^T\};\ S_O := S_O \cup \{s'\};\ Tran' := Tran' \cup \{(s^T, i, s')\}$.
   g) $W(s^T) := t_d$.
   h) For all $o \notin \texttt{out}(S_M, i)$ do
      - Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
      - $S_F := S_F \cup \{s''\};\ Tran' := Tran' \cup \{(s', o, s'')\}$.
   i) For all $o \in \texttt{out}(S_M, i)$ do
      - Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
      - $Tran' := Tran' \cup \{(s', o, s'')\}$.
      - $D' := \texttt{after}(D, i, o)$.
      - $S_{aux} := S_{aux} \cup \{(D', s'')\}$.

Fig. 5. Derivation of test cases from a specification.

is $(s, \bar{x})$. So, if $Q_1(\bar{x})$ and $Q_2(\bar{x})$ hold, we have a set with two elements, that is, $\{(s', Z_1(\bar{x}), \bar{t}_1), (s'', Z_2(\bar{x}), \bar{t}_2)\}$ where $\bar{t}_1 = (t_1, \ldots, t_n, C_1(\bar{x}))$ and $\bar{t}_2 = (t_1, \ldots, t_n, C_2(\bar{x}))$. □

Let us consider the different steps of the algorithm. The set S_{aux} initially contains a tuple with the initial situation of the specification (that is, the initial state, the initial value of variables, and the empty tuple of time values) and the initial state of the test. For each tuple belonging to S_{aux} we have two possibilities (under the heading *inductive cases*). The first possibility simply indicates that the state of the test under construction becomes a passing state (inductive case 1 of the algorithm). If the second possibility is chosen then it has to be checked that there exists a delay t_d and an input i such that the specification can perform at least one output after applying the input i after the delay t_d (this is formalized in the side condition associated with the second inductive case). If this is the case, we update the time values by considering that this delay is applied (step 2.b of the algorithm). Next, we generate an input transition in the test labelled by i and having as delay t_d (steps 2.e–g of the algorithm). Then, the whole sets of outputs is considered to generate a new transition in the test for each of these outputs. If the output is not expected by the specification (step 2.h 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. For the expected outputs (step 2.i of the algorithm) we create a transition with the corresponding output action and add the appropriate tuple to the set S_{aux}, that is, the new pending situations after traversing the transitions corresponding to the input i and each of the expected outputs.

It is important to remark that the second inductive case can be chosen only when the set S_{aux} becomes singleton. This restriction implies that our derived tests correspond to valid tests as introduced in Definition 8. Let us also note that finite tests are constructed simply by considering a step where the second inductive case is not applied.

Example 10: Next we show how our algorithm works to generate tests. In Figure 4 we present some tests derived from the specification presented in Figure 1. We suppose again that the variables of the `TEFSM` are given by a tuple $\bar{x} \in \mathbb{R}^4_+$ that initially $\bar{x} = (1, 2, 2, 1)$.

In order to generate the test T_1, a delay of 4 time units is applied in the step 2.a of the algorithm; the input i_1 is chosen. A transition labelled by this input is generated in the test. Next, all outputs are considered. Due to the fact, after i_1, that the specification only accepts the output o_2, two transitions leading to a fail state are created for the outputs o_1 and o_3 respectively (step 2.h of the algorithm). Moreover, a

transition for the output o_1 is created in the test (step 2.i of the algorithm). After this, the input i_2 is selected and, for example, a delay of 5 time units is established for this input state. After this, the corresponding transitions are created in the test for the accepted/forbidden outputs in the specification. Finally, the step 1 of the algorithm is applied in order to conclude the generation of this test. Only one pass state is created which contains a tuple of time values corresponding to the transitions traversed in the specification. It will be used to compare the time values that the implementation takes to perform outputs with the ones that are presented in the specification. The tuple $(4, 3)$ that appears in the pass state of the central branch of T_1 is extracted from the instanced temporal evolution $(4/i_1/o_2/\mathbf{4}, 5/i_2/o_1/\mathbf{3})$ of the specification that was *used* to create the test. The tests T_1 and T_2 consider the same input, i_1, in the first transition. The difference lies in the delays that have been considered for each of them, 4 and 2 time units, respectively. This fact makes that for the test T_2 the output o_1 is accepted.

Regarding T_3, let us note that the pass state presents a set with two tuples. This is due to the fact that if the input i_2 is applied without considering a delay (0 time units), the machine could trigger either the transition $s_2 \xrightarrow{i_2/o_3} s_1$ or the transition $s_2 \xrightarrow{i_2/o_3} s_5$. The machine would emit the same output o_3 but the amount of time spent in performing these transitions would be different for each of them, 4 and 2 units of time respectively. □

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 sequences that the implementation can perform. In other words, we would need a *coverage criterion* to generate a finite test suite. Since we do not assume any criteria, all we can do is to say that the derived test suite is the (possibly infinite) suite that would allow us to prove completeness. 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.

The next result relates, for a specification S and an implementation I, implementation relations and application of test suites. The non-timed aspects of our algorithm are based on the one developed for the ioco relation. So, in spite of the differences, the non-timed part of the proof of the next result is a simple adaptation of that in [34]. Regarding temporal aspects, let us remark that the existence of different instances of the same timed evolution in the specification is the reason why only some tests (and for some time values) are forced to be passed by the implementation (e.g., sometimes we only need the *fastest/slowest* test). Specifically, we take those tests matching the requirements of the specific implementation relation. In this sense, the result holds because the temporal conditions required to conform to the specification and to pass the test suite are in fact the same.

Theorem 2: Let S and I be two TEFSMs. We have that:

- $I\ \mathtt{conf}^s_a\ S$ iff I strongly passes $\mathtt{tests}(S)$ for any time.
- $I\ \mathtt{conf}^s_w\ S$ iff I strongly passes $\mathtt{tests}(S)$ in the worst time.
- $I\ \mathtt{conf}^s_b\ S$ iff I strongly passes $\mathtt{tests}(S)$ in the best time.
- $I\ \mathtt{conf}^s_{sw}\ S$ iff I strongly passes $\mathtt{tests}(S)$ sometimes in the worst time.
- $I\ \mathtt{conf}^s_{sb}\ S$ iff I strongly passes $\mathtt{tests}(S)$ sometimes in the best time.
- $I\ \mathtt{conf}^w_a\ S$ iff I weakly passes $\mathtt{tests}(S)$ for any time.
- $I\ \mathtt{conf}^w_w\ S$ iff I weakly passes $\mathtt{tests}(S)$ in the worst time.
- $I\ \mathtt{conf}^w_b\ S$ iff I weakly passes $\mathtt{tests}(S)$ in the best time.
- $I\ \mathtt{conf}^w_{sw}\ S$ iff I weakly passes $\mathtt{tests}(S)$ sometimes in the worst time.
- $I\ \mathtt{conf}^w_{sb}\ S$ iff I weakly passes $\mathtt{tests}(S)$ sometimes in the best time.

Proof: We will prove the first result. The technique is similar for the other ones.

First, let us show that I strongly passes $\mathtt{tests}(S)$ for any time implies $I\ \mathtt{conf}^s_a\ S$. We will use the contrapositive, that is, we will suppose that $I\ \mathtt{conf}^s_a\ S$ does not hold and we will prove that I does not strongly pass $\mathtt{tests}(S)$ for any time. If $I\ \mathtt{conf}^s_a\ S$ does not hold then we have two possibilities:

- Either $I\ \mathtt{conf}_f\ S$ does not hold, or
- there exists an instanced timed evolution $e = (\bar{t}, \sigma, \bar{t}_o) \in \mathtt{InsTEvol}(I)$ such that $(\bar{t}, \sigma) \in \mathtt{InsFEvol}(S)$ and $e \notin \mathtt{InsTEvol}(S)$.

Let us consider the first case, that is, we suppose that $I\ \mathtt{conf}_f\ S$ does not hold. Then, there exist $e = (\bar{t}, \sigma)$, with $\sigma = (i_1/o_1, \ldots, /i_r/o_r)$ and $\bar{t} = (t_1, \ldots, t_r)$, and $e' = (\bar{t}, \sigma')$, with $\sigma' = (i_1/o_1, \ldots, /i_r/o'_r)$, being $r \geq 1$ and such that $e \in \mathtt{InsFEvol}(S)$, $e' \in \mathtt{InsFEvol}(I)$, and $e' \notin \mathtt{InsFEvol}(S)$. We will show that there exists a test $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C, W) \in \mathtt{tests}(S)$ such that $T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$, with $s^T \in S_P$, and $T \overset{\sigma'}{\Longrightarrow}_{\bar{t}} u^T$, with $u^T \in S_F$. By constructing such a test T we obtain $I \parallel T \overset{\sigma'}{\Longrightarrow}_{\bar{t}} u^T$, for a fail state u_T. Thus, we conclude S does not pass $\mathtt{tests}(S)$. We will build this test T by applying the algorithm given in Figure 5. The different non-deterministic choices underlying the algorithm will be resolved in the following way:

1) for $1 \leq j \leq r$ do
   a) Apply inductive case 2 for the input action i_j selecting $t_d := t_j$.
   b) Apply inductive case 1 for all the elements $(D, s^T) \in S_{aux}$ that are obtained by processing an output different from o_j.

   endfor
2) Apply the first inductive case for the last element $(D, s^T) \in S_{aux}$.

Let us remark that step 1.(a) corresponds to continue the construction of the test in the state reached by the transition labelled by o_{j-1} (in the case of $j = 1$ we mean the initial state of the test). Let us also note that the *spine* of the constructed test is the sequence σ.

Since $e' \notin \mathtt{InsFEvol}(S)$, the last application of the second inductive case for the output o'_r must be necessarily associated to the step 2.(e). So, the previous algorithm generates a

test T such that $T \stackrel{\sigma'}{\Longrightarrow}_{\bar{t}} u^T$, with $u^T \in S_F$. Since $e' \in \texttt{InsFEvol}(I)$ we have that $I \| T \stackrel{\sigma'}{\Longrightarrow}_{\bar{t}} u^T$. 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 strongly pass $\texttt{tests}(S)$ for any time.

Let us suppose now that $I \, \texttt{conf}_a^s \, S$ does not hold because there exists $e = (\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$ such that $(\bar{t}, \sigma) \in \texttt{InsFEvol}(S)$ and $e \notin \texttt{InsTEvol}(S)$. Since $(\bar{t}, \sigma) \in \texttt{InsFEvol}(S)$, let us consider the test T we built before. We have again $T \stackrel{\sigma}{\Longrightarrow}_{\bar{t}} s^T$, with $s^T \in S_P$. The time stamps associated with the state s^T are generated by considering all the possible tuples of time values in which σ could be performed in S by considering the delays contained in $\bar{t}$. Besides, since $e \in \texttt{InsTEvol}(I)$, we also have $I \| T \stackrel{\sigma}{\Longrightarrow}_{\bar{t}, \bar{t}o} s^T$. Thus, if $e \notin \texttt{InsTEvol}(S)$ then $\bar{t}_o \notin C(s^T)$. We conclude I does not strongly pass $\texttt{tests}(S)$ for any time.

Let us show now that $I \, \texttt{conf}_a^s \, S$ implies I passes strongly $\texttt{tests}(S)$ for any time. Again by contrapositive, we will assume that I does not strongly pass $\texttt{tests}(S)$ for any time and we will conclude that $I \, \texttt{conf}_a^s \, S$ does not hold. If I does not strongly pass $\texttt{tests}(S)$ for any time then we have two possibilities:

- Either $\texttt{pass}(I, \texttt{tests}(S))$ does not hold, or
- there exist $e = (\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$ and $(T, s^T) \in \texttt{Test}(e, \texttt{tests}(S))$ such that $I \| T \stackrel{\sigma}{\Longrightarrow}_{\bar{t}, \bar{t}_o} s^T$, with $s^T \in S_P$ and $\bar{t}_o \notin C(s^T)$.

First, let us assume that I does not strongly pass $\texttt{tests}(S)$ for any time because $\texttt{pass}(I, \texttt{tests}(S))$ does not hold. This means that there exists a test $T \in \texttt{tests}(S)$ and some $\sigma = (i_1/o_1, \ldots, i_r/o_r)$, $s^T \in S_F$, and $\bar{t}$ such that $I \| T \stackrel{\sigma}{\Longrightarrow}_{\bar{t}} s^T$. Then, there exists $\bar{t}$ such that $T \stackrel{\sigma}{\Longrightarrow}_{\bar{t}} s^T$. According to our derivation algorithm, a branch of a derived test leads to a fail state only if its associated output action is not expected in the specification. Thus, $e = (\bar{t}, \sigma) \notin \texttt{InsFEvol}(S)$. 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 a sequence of pairs input/output $\sigma' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r)$ such that $e' = (\bar{t}, \sigma') \in \texttt{InsFEvol}(S)$. Given the fact that $e \in \texttt{InsFEvol}(I)$, $e \notin \texttt{InsFEvol}(S)$, and $e' \in \texttt{InsFEvol}(S)$, we have that $I \, \texttt{conf}_f \, S$ does not hold. We conclude $I \, \texttt{conf}_a^s \, S$ does not hold.

Let us suppose now that I does not pass $\texttt{tests}(S)$ for any time because there exist $e = (\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$ and $T \in \texttt{Test}(e, \texttt{tests}(S))$ such that $I \| T \stackrel{\sigma}{\Longrightarrow}_{\bar{t}, \bar{t}_o} s^T$, with $s^T \in S_P$ and $\bar{t}_o \notin C(s^T)$. Since $s^T \in S_P$ we deduce $(\bar{t}, \sigma) \in \texttt{InsFEvol}(S)$. Besides, by considering that $\bar{t}_o \notin C(s^T)$, we deduce $(\bar{t}, \sigma, \bar{t}_o) \notin \texttt{InsTEvol}(S)$. Finally, by using $(\bar{t}, \sigma, \bar{t}_o) \in \texttt{InsTEvol}(I)$, we conclude that $I \, \texttt{conf}_a^s \, S$ does not hold. □

As a straightforward corollary we have that the dependencies between conformance relations that we presented in Theorem 1 also hold for the corresponding testing relations. For instance, since $\texttt{conf}_a \Rightarrow \texttt{conf}_w$ we also deduce that "passes $\mathcal{T}$ at any time" implies "passes $\mathcal{T}$ in the worst time."

VI. Conclusions and Future Work

In this paper we have proposed a new model to specify timed systems. In contrast with most approaches, our formalism allows to define time in two different ways: Duration of actions and timeouts of the system. Thus, by separating these two notions, temporal properties of systems are easier to specify than by using a formalism where only one of the possibilities is available. We have also developed a testing theory. On the one hand, we have defined several conformance relations that take into account the influence of non-determinism in the behavior of systems. On the other hand, we have introduced a notion of test. In order to capture the timed behavior of the system under test, test can both delay the execution of the implementation and record the time that it took to perform a given action. Finally, we have also presented an algorithm to derive sound and complete test suites with respect to the implementation relations presented in this paper.

In terms of future work, we would like to take this work as a first step, together with [35] and [36], to define a testing theory for systems presenting stochastic time together with timeouts. An interesting continuation of this work, as suggested one of the reviews, consists in dealing with fault models. In this paper, and keeping the classical approach of ioco-like frameworks, we considered that any fault can appear. However, it would be interesting to analyze how the results of this paper have to be adapted to deal with *restricted* fault models, for example, to consider only those machines that have a (known) bound on the number of states. In this line, we also consider to study different methods for reducing the size of the generated test suite. Finally, we would like to apply our methodology to some *real* systems, in particular, to the testing of hardware circuits. We already have a first prototype that has to be optimized since it does not scale yet as to deal with systems having a big number of states.

Acknowledgments

We would like to thank the reviewers of this paper for the careful reading. The quality of the paper has notably increased by considering their useful comments and suggestions.

References

[1] M. Merayo, M. Núñez, and I. Rodríguez, "Extending EFSMs to specify and test timed systems with action durations and timeouts," in *26th IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'06, LNCS 4229*. Springer, 2006, pp. 372–387.

[2] B. Bosik and M. Uyar, "Finite state machine based formal methods in protocol conformance testing," *Computer Networks & ISDN Systems*, vol. 22, pp. 7–33, 1991.

[3] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines: A survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.

[4] A. Petrenko, "Fault model-driven test derivation from finite state models: Annotated bibliography," in *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*. Springer, 2001, pp. 196–205.

[5] E. Brinksma and J. Tretmans, "Testing transition systems: An annotated bibliography," in *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*. Springer, 2001, pp. 187–195.

[6] K. El-Fakih, N. Yevtushenko, and G. v. Bochmann, "FSM-based incremental conformance testing methods," *IEEE Transactions on Software Engineering*, vol. 30, no. 7, pp. 425–436, 2004.

[7] I. Rodríguez, M. Merayo, and M. Núñez, "$\mathcal{HOTL}$: Hypotheses and observations testing logic," *Journal of Logic and Algebraic Programming (in press)*, 2007.

[8] J. Sifakis, "Use of Petri nets for performance evaluation," in *3rd Int. Symposium on Measuring, Modelling and Evaluating Computer Systems*. North-Holland, 1977, pp. 75–93.

[9] G. Reed and A. Roscoe, "A timed model for communicating sequential processes," *Theoretical Computer Science*, vol. 58, pp. 249–261, 1988.

[10] W. Yi, "CCS+ Time = an interleaving model for real time systems," in *18th Int. Colloquium on Automata, Languages and Programming, ICALP'91, LNCS 510*. Springer, 1991, pp. 217–228.

[11] X. Nicollin and J. Sifakis, "An overview and synthesis on timed process algebras," in *3rd Int. Conf. on Computer Aided Verification, CAV'91, LNCS 575*. Springer, 1991, pp. 376–398.

[12] J. Quemada, D. d. Frutos, and A. Azcorra, "TIC: A TImed Calculus," *Formal Aspects of Computing*, vol. 5, pp. 224–252, 1993.

[13] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[14] M. Hennessy and T. Regan, "A process algebra for timed systems," *Information and Computation*, vol. 117, no. 2, pp. 221–239, 1995.

[15] J. Davies and S. Schneider, "A brief history of timed CSP," *Theoretical Computer Science*, vol. 138, pp. 243–271, 1995.

[16] J. Baeten and C. Middelburg, *Process algebra with timing*, ser. EATCS Monograph. Springer, 2002.

[17] 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*. Springer, 2006, pp. 103–117.

[18] J. Springintveld, F. Vaandrager, and P. D'Argenio, "Testing timed automata," *Theoretical Computer Science*, vol. 254, no. 1-2, pp. 225–257, 2001, previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.

[19] R. Barbuti and L. Tesei, "Timed automata with urgent transitions," *Acta Informatica*, vol. 40, no. 5, pp. 317–347, 2004.

[20] B. Gebremichael and F. Vaandrager, "Specifying urgency in timed I/O automata," in *3rd IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM 2005)*. IEEE Computer Society Press, 2005, pp. 64–73.

[21] P.-A. Hsiung, S.-W. Lin, Y.-R. Chen, C.-H. Huang, J.-J. Yeh, H.-Y. Sun, C.-S. Lin, and H.-W. Liao, "Model checking timed systems with urgencies," in *4th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'06, LNCS 4218*. Springer, 2006, pp. 67–81.

[22] 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*. IEEE Computer Society Press, 1997, pp. 199–206.

[23] 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*. Kluwer Academic Publishers, 1999, pp. 197–214.

[24] 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*. Springer, 2003, pp. 211–225.

[25] M. Krichen and S. Tripakis, "Black-box conformance testing for real-time systems," in *11th Int. SPIN Workshop on Model Checking of Software, SPIN'04, LNCS 2989*. Springer, 2004, pp. 109–126.

[26] 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*. Springer, 2005, pp. 264–279.

[27] 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.

[28] R. Cardell-Oliver, "Conformance tests for real-time systems with timed automata specifications," *Formal Aspects of Computing*, vol. 12, no. 5, pp. 350–371, 2000.

[29] 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*. Springer, 1998, pp. 251–260.

[30] D. Mandrioli, S. Morasca, and A. Morzenti, "Generating test cases for real time systems from logic specifications," *ACM Transactions on Computer Systems*, vol. 13, no. 4, pp. 356–398, 1995.

[31] J. Peleska and M. Siegel, "Test automation of safety-critical reactive systems," *South African Computer Journal*, vol. 19, pp. 53–77, 1997.

[32] 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*. Springer, 2002, pp. 1–16.

[33] A. Petrenko and N. Yevtushenko, "Conformance tests as checking experiments for partial nondeterministic FSM," in *5th Int. Workshop on Formal Approaches to Software Testing, FATES'05, LNCS 3997*. Springer, 2006, pp. 118–133.

[34] J. Tretmans, "Test generation with inputs, outputs and repetitive quiescence," *Software – Concepts and Tools*, vol. 17, no. 3, pp. 103–120, 1996.

[35] 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*. Springer, 2003, pp. 335–350.

[36] M. Merayo, M. Núñez, and I. Rodríguez, "Implementation relations for stochastic finite state machines," in *3rd European Performance Engineering Workshop, EPEW'06, LNCS 3964*. Springer, 2006, pp. 123–137.

Mercedes G. Merayo is a Ph.D. student in the Computer Systems and Computation Department of Universidad Complutense de Madrid, Spain. She also holds an assistant teacher position at the same department. She has published 15 papers in refereed journals and international venues. Her research interests are formal methods and probabilistic/timed/stochastic extensions in formal testing.

Manuel Núñez received a Ph.D. in Mathematics/Computer Science in 1996 and a MS degree in Economics in 2002, in both cases from Universidad Complutense de Madrid, Spain. Since 1997, he is an associate professor in the Computer Systems and Computation Department at the same university. He has published more than 80 papers in refereed journals and international venues. He regularly serves in the Program Committee of conferences such as Forte and TestCom, being the co-chair of Forte 2004 and FATES 2006, among others. His research interests include formal notions of testing and performance evaluation, application of economic theory to computer science, and formal specification and analysis of e-commerce systems.

Ismael Rodríguez received his Ph.D. in Computer Science from Universidad Complutense de Madrid, Spain, in 2004, receiving the Best Thesis Award in the Computer Science School. Since 2007, he is an associate professor in the Computer Systems and Computation Department at the same university. He has published more than 50 papers in refereed journals and international venues. His research interests include formal methods, testing techniques, e-learning environments and e-commerce.