SOFTWARE TESTING, VERIFICATION AND RELIABILITY
*Softw. Test. Verif. Reliab.* 2010; **00**:1–26
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/stvr

# A formal framework to test soft and hard deadlines in timed systems

Mercedes G. Merayo, Manuel Núñez, and Ismael Rodríguez *

*Departamento Sistemas Informáticos y Computación, Universidad Complutense de Madrid, 28040 Madrid, Spain*

SUMMARY

This paper introduces a formal framework to specify and test systems presenting both soft and hard deadlines. While hard deadlines must be always met on time, soft deadlines can be sometimes met in a different time, usually greater, from the specified one. It is this characteristic (to formally define *sometimes*) what produces several reasonable alternatives to define appropriate *implementation relations*, that is, relations to decide whether an implementation is correct with respect to a specification. In addition to introduce these relations, the paper also presents a formal testing framework to test implementations and provides an algorithm to derive sound and complete test suites with respect to the implementation relations previously defined. That is, an implementation conforms to a specification if and only if the implementation successfully passes all the tests belonging to the suite derived from the specification. Copyright © 2010 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Formal testing techniques; timed systems; soft and hard deadlines

## 1. INTRODUCTION

The complexity of computational systems has reached a level where old intuition-based methods to check the reliability of systems are no longer valid. Hundreds of system components may be built by different parties and no team member is able to give a precise prediction about where critical errors could appear. In these situations, it is more suitable to use automatic tools to analyze systems according to a given *formal method*. These methods check system properties by systematically confronting the ideal system definition or its practical realization against a big (ideally, complete) battery of critical situations. In this way, it is possible to check a property in a model representing the behavior of the system (model checking [1, 2]) or to apply tests to an implementation, trying to check whether it fulfills a property (testing techniques [3, 4]). If testing is supported by a specification of the system then the approach is usually called formal testing or model-based testing [5, 6, 7, 8]. In both cases, model checking and formal testing, a model representing the ideal system behavior is required. Even though the initial frameworks to formally specify systems, such as Process Algebras [9] and Petri Nets [10, 11, 12], focused on defining *what* a system must do, a new generation of models allow the users to define *how* the system performs these actions. In this way, probabilistic and temporal issues were introduced in specification formalisms [13, 14, 15, 16, 17, 18, 19, 20].

*E-mails: mgmerayo@fdi.ucm.es, mn@sip.ucm.es, isrodrig@sip.ucm.es

Contract/grant sponsor: The Spanish MEC project TESIS and the UCM-BSCH program to fund research groups; contract/grant number: TIN2009-14312-C02-01 and GR58/08 - group number 910606.

Formal testing went through a similar process: From testing only *what* to also test *how*. 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, as in the case of model definition, formal testing techniques are also dealing with *non-functional* properties such as the probability of an event to happen or the time that it takes to perform a certain action (for example, [21, 22, 23, 24, 25, 26, 27, 28]).

One of the problems when specifying timed systems is that it is not always easy to precisely establish the time bounds associated with the tasks that the system performs. Thus, it is sometimes useful to consider some degree of *indecision* in such specifications. In this line, formalisms to represent systems presenting *stochastic-time* behavior [29, 30, 31, 32, 33]) can specify constraints such as "with probability $p$, the task must finish before $t$ time units have passed". So, the specifier does not need to provide the precise point of time associated with a task, but a probabilistic estimation of the time value(s). However, there are situations where the specifier either does not have such probabilistic information or does not want to provide such information because it might unnecessarily complicate the model. In this case, the most appropriate way to specify time constraints is to use *time intervals*, that is, the specifier provides a set of possible time values, instead of just one, but without quantifying the probability that each value of the interval has. Moreover, it may happen that while testing the correctness of a system, the tester can accept some *imprecision* in the temporal behavior of the system. For example, if the specifier cannot precisely define the temporal constraints of a system, the tester can also have problems to determine what is the exact notion of passing a test *on time*. Moreover, it can be admissible that the execution of a task sometimes lasts longer than expected: If most of the times the task is performed on time, a couple of delays can be tolerated. This is the idea of a *soft* deadline, in contrast with *hard* deadlines that have to be always met on time. Finally, another reason for the tester to accept imprecisions is that the artifacts measuring time while testing a system could not be as precise as desirable. In this case, an apparent wrong behavior due to bad timing can be in fact correct since it may happen that the *clocks* are not working properly.

This paper proposes a novel formal framework to specify and test systems where time considerations fall in the cases previously commented. Time will be introduced in specifications by extending classical finite state machines with time intervals associated to the performance of actions. Intuitively, a transition of a finite state machine such as $s \xrightarrow{i/o} s'$ indicates that if the machine is in a state $s$ and receives and input $i$ then it will produce and output $o$ and it will change its state to $s'$. In the timed extension of finite state machines used in this paper, a transition such as $s \xrightarrow{i/o}_{[t_1,t_2]} s'$ 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 $t_2$ between the reception of the input and the generation of the output. The main weakness of this model is that its expressive power is limited when compared with other more complex formalisms such as timed automata [34]. Most notably, it is not possible to express relations between timed behavior in different states of the machine. This can be easily done with timed automata by defining constraints involving different clocks that are initialized at different points. In order to reach an expressive power similar to timed automata, finite state machines should be augmented with data, as well as with conditions on data to choose among transitions, in a way similar to the extension considered in the paper [35]. Therefore, this model can be considered as the basis for a more expressive model. On the positive side, the simplicity of this formalism is also its main advantage: It is really easy to define reactive systems where time has to be taken into account but where there is no need to express complex timed constraints, that is, where the expressive power of timed automata is not fully used. More precisely, the systems that can be modeled using the formalism presented in this paper do not have infinite data domains (finite domains can be deployed as usual by generating a transition for each value of the domain), they require only one clock, and this clock is initialized before each transition is performed. There is a wide variety of systems that stick to these constraints. For example, ovens [36] or washing machines (as used in this paper) can be modeled and analyzed



DOI: 10.1002/stvr

with this type of formalism (one clock and finite domains). It is important to emphasize that the simplicity of the formalism allows non-experts in formal methods, but with a background on reactive systems that receive inputs and produce outputs, to provide a first draft (as the experience shows, very accurate) of a formal specification of a part of the system or functionality that needs to be tested.

Testing, as well as the definition of implementation relations, will depend on measuring time values and accepting the performance of the system if the time behavior is correct *up to* an admissible error. The possible definition of *admissible* will give raise to several alternative implementation relations and several notions of passing a test. However, there is still a last issue when dealing with systems where time requirements are given by means of intervals. Since a black-box testing framework is assumed, it is not possible to check that the intervals governing the behavior of the implementation are correctly related with the ones corresponding to the implementation. In fact, the execution of a test will return the time associated with the performance of transitions, not the associated time interval. As a consequence, all that can be achieved by testing is to *approximately* infer the actual intervals of the implementation.

The main original point of this paper is the presentation of a novel specification formalism and a testing methodology, including the definition of implementation relations, to deal with systems where imprecisions in the analysis of time behaviors are allowed. This is a major difference with respect to the usual approaches where a system must always present an appropriate time behavior to be correct. In contrast with other approaches to non-strict requirements that will be analyzed in Section 6, devoted to related work, this testing framework provides a rich way to express what is *allowed*, *partially allowed*, or *completely forbidden* in non-strict (temporal) requirements. In fact, the proposed implementation relations allow the tester to explicitly consider, for each system action,

- an *interval* where observed times will be considered as correct;
- a temporal area where any observation will make that the system is considered as incorrect (even if a single observation falls within it); and
- an undesirable (but partially acceptable) area where observations will be accepted to some extent, explicitly defined by the tester.

By making a suitable definition of allowed/forbidden areas for each task that the system can perform, the tester has the capability to impose different non-strict requirements to each system activity. Moreover, the tester is able to decide *how bad* observations in the third area are since she may consider either that the *proportion* of all observations falling within this area determines the system (in-)correctness, or that the *distance* of each observation within this area to the allowed one must be considered as well. This paper represents an extended and revised version of the paper [37]. This paper adds detailed discussions and explanations of the main concepts, new illustrative examples, both numerical and practical, proofs of selected results, and a test derivation algorithm to generate sound and complete test suites.

The rest of the paper is organized as follows. Section 2 introduces a notion of timed finite state machine and gives some auxiliary notation. Section 3 presents several timed conformance relations. Section 4 shows how tests are defined and applied to implementations. Section 5 gives an algorithm to generate sound and complete test suites. Related work is reviewed and briefly compared with the approach presented in this paper in Section 6. Finally, Section 7 gives the conclusions and some lines for future work.

## 2. EXTENDING FINITE STATE MACHINES WITH TIME INTERVALS

This section introduces a notion of timed finite state machine, called `IFSM`, and some concepts that will be used along the paper. The main difference with respect to usual `FSM`s consists in the addition of *time* to indicate the lapse between offering an input and receiving an output. First, notation related to time intervals, sets, and multisets is presented.

*Definition 1*
Let $a_1 \in \mathbf{R}_+$, $a_2 \in \mathbf{R}_+ \cup \{\infty\}$, and $a_1 \leq a_2$. Then, $d = [a_1, a_2]$ is a *time interval* and $\pi_i(d)$, for



DOI: 10.1002/stvr

$i \in \{1,2\}$, denotes the value $a_i$. From now on, assume that for all $r \in \mathbf{R}_+$, $r < \infty$, $r + \infty = \infty$, and $\frac{r}{\infty} = 0$. The set of time intervals is denoted by $\mathcal{I}_{\mathbb{R}_+}$.

Given two time intervals $d_1 = [a_{11}, a_{12}]$ and $d_2 = [a_{21}, a_{22}]$, $d_1 + d_2$ denotes the time interval $[a_{11} + a_{21}, a_{12} + a_{22}]$. The addition of time intervals can be generalized to $n$ summands in the expected way. Given $n$ time intervals $d_1 = [a_{11}, a_{12}], \ldots, d_n = [a_{n1}, a_{n2}]$, $\sum d_i$ denotes the time interval $[\sum a_{i1}, \sum a_{i2}]$.

Given a set $S$, $|S|$ denotes the cardinal of $S$, $\mathcal{P}(S)$ denotes the powerset of $S$, and $\wp(S)$ denotes the powermultiset of $S$, that is, the set of multisets conformed from elements belonging to $S$. The symbols $\{\!\!\{$ and $\}\!\!\}$ will be used to denote multisets. Given a multiset $\mathcal{M}$, $r \in \mathcal{M}$ denotes that $r$ appears in $\mathcal{M}$ (that is, $r$ has multiplicity greater than 0). $\|\mathcal{M}\|$ denotes the cardinal of $\mathcal{M}$ including multiplicity of its elements. For example, $\|\{\!\!\{1,2,3,1,2\}\!\!\}\| = 5$. □

A temporal requirement such as $[t_1, t_2]$ indicates that the associated task should take at least $t_1$ time units and at most $t_2$ units to be performed. Intervals like $[0, t_2]$, $[t_1, \infty]$, or $[0, \infty]$ denote the absence of a temporal lower/upper bound and the absence of any bound, respectively. In the case of intervals such as $[t, \infty]$, there is an abuse of notation since this interval represents, in fact, the interval $[t, \infty)$.

*Definition 2*

An *Interval Finite State Machine*, in the following `IFSM`, 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, and $d \in \mathcal{I}_{\mathbb{R}_+}$ is the time interval associated with the transition.

The `IFSM` $M$ is *input-enabled* if for all states $s \in S$ and inputs $i \in I$, there exist $s'$, $o$, and $d$ such that $(s, s', i, o, d) \in Tr$. The `IFSM` $M$ is *observable* if there do not exist two different transitions $(s, s_1, i, o, d_1), (s, s_2, i, o, d_2) \in Tr$. □

Intuitively, a transition $(s, s', i, o, d)$ indicates that if the machine is in state $s$ and receives the input $i$ then, after a time belonging to the interval $d$ has passed, the machine emits the output $o$ and moves to $s'$. It is important to remark that these machines can patiently wait in a state until they receive an input. The amount of time elapsed during this waiting period will not influence the behavior of the machine. For example, the traces of a machine do not consider this waiting time. An alternative possibility consists in assuming that the environment is *fast* enough or that the inputs are provided beforehand so that there is no delay between the production of an output and the application of the next input.

In this paper, the systems under test must be *input-enabled*. This notion needs and additional explanation since, in this paper, it slightly departs from its standard meaning. In a non-timed setting, a system is input-enabled if "for all states of the system, inputs are available." If this notion is adapted to a timed framework, the natural translation would be "for all states of the system, inputs are available at all times." The notion used in this paper differs from this one since inputs are available only when they can be performed by the corresponding machine. In other words, inputs are not available during the lapse between an input is received and the corresponding output is emitted. Note that this notion of input-enabledness is indeed realistic. On the one hand, if a testing approach where inputs are provided beforehand is used then, in the formalism presented in this paper, it does not matter the actual time when they are applied (it is only important to apply them in the same order provided by the tester). Therefore, it is not relevant that the machine is not accepting inputs while completing its transitions. On the other hand, if inputs are provided by the tester in an adaptive way (that is, according to the reactions previously observed) then this approach can be implemented in the general testing scenario used in [38] by considering that these machines are equipped with a *green light*. When this light is on, the user knows that the system is performing some internal activities (the ones producing the output associated with the provided input) and that it cannot accept any input. Therefore, the user precisely knows when to apply inputs.



DOI: 10.1002/stvr

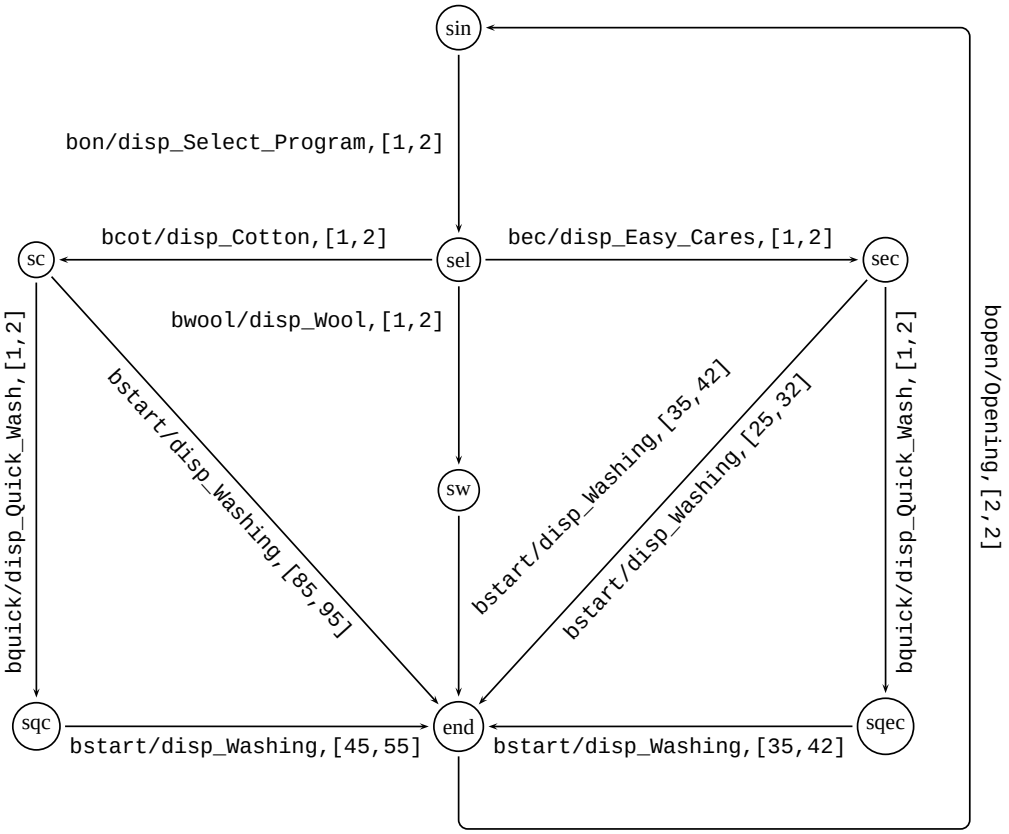



Figure 1. An example of Interval Finite State Machine.

*Example 1*

Consider the graphical representation of an `IFSM` given in Figure 1. This system will be used as a running example to illustrate the main concepts of the paper. This relatively simple system adequately represents the kind of systems covered by the current framework: Systems whose evolutions can be described as an alternating sequence of inputs and outputs and where time plays an important role, but without time conditions affecting different parts of the system. The machine depicted in Figure 1 is based on a simplification of a washing machine.

The washing machine provides a set of buttons (`bon`, `bwool`, `bcot`, `bec`, `bquick`, `bstart`, `bopen`) and a screen. The buttons allow the user to access the different functionalities of the machine and correspond to the inputs that can be applied. In this system, some outputs correspond to the display of different *messages*. An output such as `disp_m` indicates that `m` is displayed on the screen. During the rest of the paper, and in the context of this running example, `m` will be used as a shorthand of `disp_m`. The specification has eight states, being `sin` its initial state. The different transitions are labeled by the pressed button (the input of the transition), the action that will be executed considering the input applied (the output of the transition), and a time interval representing the lower and upper bounds of time that the washing machine can spend between the moment the button is pressed and the associated message/action is displayed/executed.

The behavior of the washing machine is as follows. The initial state, `sin`, corresponds to the point in which the user switches on the machine and the system moves to state `sel`. At this state the user can choose among the different programs available by pressing the buttons `bcot` (cotton), `bec` (easy cares) or `bwool` (woolens). This interaction produces the display of the selected program, changing the system to the `sc`, `sec` or `sw` state, respectively. At states `sc` and `sec` the button



DOI: 10.1002/stvr

bquick can be pressed to choose a short washing cycle. If this option is not selected, then the machine will perform a complete cycle when the button bstart is pressed. In the case that the user chooses the program for woolens, the quick cycle cannot be selected. Once the washing has finished, the user must press the bopen button in order to unlock the door. After two units of time the door will be released.

Overall, the following processes are considered in the system:

- The user chooses the program: Cotton, Woolens or Easy Cares.
- The user selects, by pressing the button bquick, a short washing cycle. It can be selected only if the program Cotton or Easy Cares was chosen.
- The user must press the button bopen to unblock the door and switch off the machine.

The machine patiently waits at state sel until it receives either bcot, bec or bwool. Note that the amount of time that the machine stayed at state sel will not be relevant in its subsequent behavior. If, for example, the user eventually presses the button bcot then it will produce the output disp_Cotton, taking between one and two time units, and it will move to state sc. As explained before, the machine will not be able to process an input until the previous output is performed. For example, in the previously described situation where bcot is received, the system will not be able to process any input from the moment it receives bcot until it displays the message disp_Cotton. □

A more *operational* view of IFSMs can be given by unfolding the time values contained in the corresponding time intervals to generate a timed labeled transition system.

*Definition 3*
Let $M = (S, I, O, Tr, s_{in})$ be an IFSM. The *equivalent timed labeled transition system* of $M$ is a tuple $(S, I \times O, Tr', s_{in})$, where $Tr' \subseteq S \times (I \times O) \times \mathbf{R}_+ \times S$ is defined such that $(s, (i, o), t, s') \in Tr'$ if and only if there exists $d \in \mathcal{I}_{\mathbb{R}_+}$ such that $(s, s', i, o, d) \in Tr$ and $t \in d$. □

This operational definition is given only to provide an encoding of IFSMs into a *standard* model to represent timed systems but it will not be used along the paper since it gives raise to an infinite number of transitions.

*Example 2*
Consider the specification presented in Example 1. The equivalent timed labeled transition system has indeed the same states as the original specification, with the same initial state, and its alphabet is given by the product of the set of inputs by the set of outputs. Regarding the transitions of the labeled transition system, each original transition produces a set of transitions, one transition for each value of the associated interval. For example, consider the transition of the original specification that outgoes from state sqec, reaches state end, and it is labeled with the pair bstart/disp_Washing and the interval $[35, 42]$. This transition generates a transition (sqec,(bstart,disp_Washing),$t$,end) for each value $t$ belonging to the interval $[35, 42]$. □

As usual, a *trace* is a sequence of input/output pairs together with a time interval. The time interval indicates the lower and upper bounds associated with the lapses between receiving each input and performing its associated output. Therefore, this interval will be computed by the addition of all the time intervals associated with the transitions conforming the trace. An *evolution* is a trace starting at the initial state of the machine.

*Definition 4*
Let $M = (S, I, O, Tr, s_{in})$ be an IFSM. A *timed trace*, or simply *trace*, of $M$ is a tuple $(s, s', (i_1/o_1, \ldots, i_r/o_r), d)$ if there exist transitions $(s_1, s_2, i_1, o_1, d_1), \ldots, (s_r, s_{r+1}, i_r, o_r, d_r) \in Tr$ such that $s = s_1$, $s' = s_{r+1}$, and $d = \sum d_i$.

The sequence $(i_1/o_1, \ldots, i_r/o_r)$ is a *non-timed evolution*, or simply *evolution*, of $M$ if $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), d)$ is a trace of $M$ for some $d \in \mathcal{I}_{\mathbf{R}_+}$ and $s' \in S$. NTEvol$(M)$ denotes the set of non-timed evolutions of $M$.

The pair $((i_1/o_1, \ldots, i_r/o_r), d)$ is a *timed evolution* of $M$ if there exists $s' \in S$ such that $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), d)$ is a trace of $M$. TEvol$(M)$ denotes the set of timed evolutions of $M$.



DOI: 10.1002/stvr

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

$$\texttt{setout}(M, i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n) = \left\{ o \in O \;\middle|\; \begin{array}{l} \exists s, s' \in S, d, d' \in \mathcal{I}_{\mathbb{R}_+} : \\ \quad (s_{in}, s, (i_1/o_1, \ldots, i_{n-1}/o_{n-1}), d) \text{ trace of } M \\ \quad \wedge\ (s, s', i_n, o, d') \in Tr \end{array} \right\}$$

□

The notion of timed trace/evolution is important for the subsequent theory and deserves some additional discussion. Specifically, there is an alternative way to define the time information associated with a timed evolution. It consists in separately recording the time associated with each component of the trace, so that there is not a unique time interval associated with the whole sequence but a time interval associated with each input/output pair. This approach would give rise to implementation relations with a higher distinguishing power. Note that it is very easy to adapt the current framework to deal with this type of timed evolutions. In fact, it would be enough to consider the notions introduced in a previous work by the authors [35] so that the current notion of timed evolution would produce *weak* implementation relations while the alternative notion would produce *strong* relations.

*Example 3*
Consider again the specification of a washing machine presented in Figure 1. The tuple

```
(sin,sqc,(bon/Select_Program,bcot/Cotton,bquick/Quick_Wash),[3,6])
```

is a trace of $M$. Its associated timed evolution is

```
((bon/Select_Program,bcot/Cotton,bquick/Quick_Wash),[3,6])
```

This timed evolution represents that from state `sin` the machine will perform the sequence of displays `Select_Program,Cotton,Quick_Wash` after the user has pressed the sequence of buttons `bon,bcot,bquick`, and the system will reach the state `sqc`. The associated interval indicates that the addition of the three lapses of time recorded between receiving each input and performing the associated output must belong to $[3, 6]$. As it has been already mentioned, the time that the system is waiting to receive the corresponding inputs is not taken into account.

If the previously discussed notion of timed evolution were used, then it would become

```
(bon/Select_Program/[1,2],bcot/Cotton/[1,2],bquick/Quick_Wash/[1,2])
```

□

## 3. IMPLEMENTATION RELATIONS

This section introduces several implementation relations. Following the classical pattern, an implementation *conforms* to a specification if for all possible sequences of inputs that the specification can perform, the outputs emitted by the implementation are a subset of those of the specification. Intuitively, this means that the implementation cannot *invent* a behavior (that is, an output) for those traces that the specification can perform. This pattern is borrowed from `ioco` [39] and was firstly introduced in the context of finite state machines in the work [40].

A specification is an `IFSM` while implementations are input-enabled `IFSM`s. This is a usual condition to assure that the implementation will react (somehow) to any input. In order to simplify the presentation, it will be assumed that specifications and implementations are given by observable `IFSM`s (see Definition 2). Note that even restricting to this kind of machines, it is still possible to 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, some degree of non-determinism is allowed within these constraints.



DOI: 10.1002/stvr

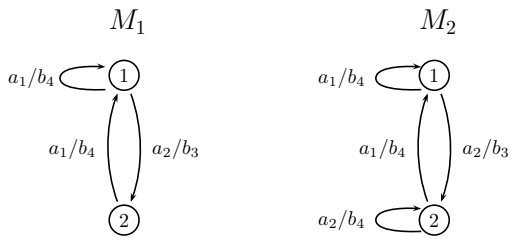



Figure 2. Examples of non-timely conformance.

*Definition 5*
Let $S$ and $I$ be two *observable* IFSMs where $I$ is also *input-enabled*. $I$ *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 $r \geq 1$, and all $e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r)$,

$$e' \in \mathtt{NTEvol}(I) \Longrightarrow e' \in \mathtt{NTEvol}(S)$$

□

From now on, it will be usually omitted that specifications and implementations are observable and that implementations are input-enabled. Note that, in the previous definition, if the specification would also have the property of being input-enabled then 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 $r \geq 1$ " can be removed, so that the notion of conformance becomes trace inclusion.

*Example 4*
Consider the systems $M_1$ and $M_2$ depicted in Figure 2, where time information has been omitted. It is straightforward to check that $M_2$ $\mathtt{conf}_{nt}$ $M_1$. Note that the non-timed evolutions of $M_2$ having as prefix the sequence $(a_2/b_3, a_2/b_4)$ are not checked because $M_1$ (playing the role of specification) cannot perform those evolutions.

Consider that $M_1$ is extended with the transition $(2, 2, a_2, b_5, d)$ so that $M_1$ is input-enabled. Then, $M_1$ does not conform to $M_2$. For example, $M_2$ may perform the non-timed evolution $e = (a_2/b_3, a_2/b_4)$, $M_1$ has the non-timed evolution $e' = (a_2/b_3, a_2/b_5)$, but $e'$ does not belong to the set of non-timed evolutions of $M_2$. Note that $e$ and $e'$ share the common prefix $a_2/b_3, a_2$. □

Since the previously introduced implementation relation will be used 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 specification $S$ can perform, that is, it is enough to consider only those sequences $e$ such that $\mathtt{setout}(S, e) \neq \emptyset$.

*Lemma 1*
Let $S$ and $I$ be two IFSMs. $I$ $\mathtt{conf}_{nt}$ $S$ if and only if for all sequences of input/outputs $i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n$, with $\mathtt{setout}(S, i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n) \neq \emptyset$, the following inclusion holds:

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

□

The following definition introduces the first timed implementation relation. In addition to the non-timed conformance of the implementation, it requires a time condition to hold: The time intervals of the implementation correspond to those of the specification.



DOI: 10.1002/stvr

*Definition 6*
Let $I$ and $S$ be IFSMs. $I$ *conforms in time* to $S$, denoted by $I\ \mathtt{conf}_{int}\ S$, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \mathtt{NTEvol}(I) \cap \mathtt{NTEvol}(S)$ and all time intervals $d \in \mathcal{I}_{\mathbb{R}_+}$,

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

□

Despite its neat definition, this relation suffers from practical problems due to the assumption that the implementation under test is a black box. The assumed black-box testing framework forbids to check whether the corresponding intervals coincide indeed. Each observation of the implementation provides *one* of the time values allowed by the implementation at a given configuration (i.e., it provides a single time within the corresponding interval of the implementation). Hence, some of the time values allowed by the implementation at this configuration could remain unknown after observing this point of the implementation any given number of times. For example, suppose that a test is applied an arbitrarily big number of times and all observed time values belong to the interval $[2,4]$. Still, the corresponding implementation interval could be $[1,5]$. Thus, it is necessary to give more *realistic* implementation relations that are less accurate but are *checkable*. The only additional assumption needed in the testing framework to capture this idea is that the time that the implementation takes to perform a given sequence can be recorded. In order to do that, the concept of *timed execution* is used. A timed execution is simply an input/output sequence together with the time that it takes to perform the sequence. In a certain sense, timed executions can be seen as *instances* of timed evolutions where the time interval is substituted by a specific time within it. Timed executions are grouped into *multisets*. These multisets will be later used to denote the observations collected by *testing* an implementation.

*Definition 7*
Let $M = (S, I, O, Tr, s_{in})$ be an IFSM. A multiset $H$ of elements contained in $(I \times O)^* \times \mathbf{R}_+$ is a *multiset of timed executions* of $M$ if for all $(e,t) \in H$ there exists $d \in \mathcal{I}_{\mathbb{R}_+}$ such that $(e,d) \in \mathtt{TEvol}(M)$ and $t \in d$.

Let $H = \{\!\{(e'_1,t_1),\ldots,(e'_n,t_n)\}\!\}$ be a multiset of timed executions of $M$ and $\Phi = \{e \mid \exists\, t : (e,t) \in H\}$ be the set of input/output sequences in $H$. The function $\mathtt{Obs\_Time}_H : \Phi \longrightarrow \wp(\mathbf{R}_+)$ defined, for all $e \in \Phi$, as $\mathtt{Obs\_Time}_H(e) = \{\!\{t \mid (e,t) \in H\}\!\}$ is called the *time function* of $H$. □

*Example 5*
Consider again the system introduced in Figure 1. The following is a possible set of timed executions

$$H = \left\{\!\!\left\{ \begin{array}{l} \mathtt{((bon/Select\_Program,bcot/Cotton,bquick/Quick\_Wash,bstart/Washing),53)} \\ \mathtt{((bon/Select\_Program,bcot/Cotton,bquick/Quick\_Wash,bstart/Washing),55)} \\ \mathtt{((bon/Select\_Program,bcot/Cotton,bquick/Quick\_Wash,bstart/Washing),55)} \\ \mathtt{((bon/Select\_Program,bcot/Cotton,bstart/Washing),87)} \\ \mathtt{((bon/Select\_Program,bcot/Cotton,bstart/Washing),88)} \\ \mathtt{((bon/Select\_Program,bcot/Cotton,bstart/Washing),88)} \\ \mathtt{((bon/Select\_Program,bcot/Cotton,bstart/Washing),89)} \\ \mathtt{((bon/Select\_Program,bwool/Wool,bstart/Washing),39)} \\ \mathtt{((bon/Select\_Program,bwool/Wool,bstart/Washing),39)} \\ \mathtt{((bon/Select\_Program,bwool/Wool,bstart/Washing),42)} \\ \mathtt{((bon/Select\_Program,bwool/Wool,bstart/Washing),40)} \end{array} \right\}\!\!\right\}$$

For example, if $e = \mathtt{(bon/Select\_Program,bwool/Wool,bstart/Washing)}$ then $\mathtt{Obs\_Time}_H(e) = \{\!\{39,39,42,40\}\!\}$. □

The following definition introduces several conformance relations where the observed time values are contrasted, in each case, with the appropriate time intervals. The purpose of this paper is to introduce implementation relations where the time behavior of the implementation does not exactly correspond to the expected one, that is, it partially deviates from the behavior defined in the specification. Intuitively, it is necessary to determine whether the amount of *incorrect* time values is



DOI: 10.1002/stvr

relevant to ensure the possible conformance of the implementation to the specification. Moreover, the deviation of the observed time values with respect to the interval can be also quantified.

The next definition introduces notation to relate a set of observed time values and a time interval. In these notions, the parameter $\alpha$ denotes the *tolerance* to have unexpected values. In particular, greater values of $\alpha$ denote smaller tolerance. The first relation, $\subseteq_\alpha$, reflects that the number of values outside the considered interval is not large. There is no distinction between values being smaller/greater than the lower/upper bound of the interval. The second relation, $\preceq_\alpha$, is used to indicate that values greater than the upper bound are not allowed and that the number of values smaller than the lower bound is acceptable. Finally, $\ll_\alpha$ is useful in situations where most of the values have to be smaller than the *lower* bound of the interval, while values greater than the upper bound are again not allowed. That is, $\ll_\alpha$ reinterprets temporal requirements imposed by intervals, stating that time values under the lower bound are always allowed, time values between the lower and the upper bound are acceptable only up to some extent (given by $\alpha$, which here imposes a requirement over a different temporal region), and time values over the upper bound are forbidden. This last relation requires that the amount of time that the system spends to perform the actions is smaller than the one specified in the corresponding transitions. The previous relations count the number of errors but do not quantify how big these errors are. Some *distance* functions allow to measure the error degree of wrong values. The first one, `dist`, considers both time values greater and smaller than the bounds of the interval. The `from_up` function considers as wrong only those values greater than the upper bound of the interval. Finally, the `from_low` function measures the values that are not fast enough, that is, such that they are greater than the *lower* bound of the interval. Thus, similarly to $\ll_\alpha$, the `from_low` relation also reinterprets the meaning of time intervals, stating that execution times over *lower* bounds are acceptable only up to some extent.

*Definition 8*
Let $d = [a_1, a_2] \in \mathcal{I}_{\mathbb{R}_+}$ be a time interval, $\mathcal{R}$ be a non-empty finite multiset of non-negative real numbers, and $0 \leq \alpha \leq 1$.

- $\mathcal{R} \subseteq_\alpha d$ holds if
$$\frac{\|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ (r < a_1 \ \vee \ r > a_2)\}\!\}\|}{\|\mathcal{R}\|} \leq 1 - \alpha$$
- $\mathcal{R} \preceq_\alpha d$ holds if
$$\frac{\|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ r < a_1\}\!\}\|}{\|\mathcal{R}\|} \leq 1 - \alpha \text{ and } \|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ r > a_2\}\!\}\| = 0$$
- $\mathcal{R} \ll_\alpha d$ holds if
$$\frac{\|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ a_1 < r \leq a_2\}\!\}\|}{\|\mathcal{R}\|} \leq 1 - \alpha \text{ and } \|\{\!\{r \mid r \in \mathcal{R} \ \wedge \ r > a_2\}\!\}\| = 0$$
- Given an observed time value $r \in \mathbf{R}_+$ and a time interval $d = [a_1, a_2] \in \mathcal{I}_{\mathbb{R}_+}$, the following three functions, and their generalization to sets of values, define *distances* from the time value(s) to the time interval:
$$\texttt{dist}(r,d) = \begin{cases} 0 & \text{if } r \in d \\ r - a_2 & \text{if } r > a_2 \\ a_1 - r & \text{if } r < a_1 \end{cases} \qquad \texttt{dist}(C,d) = \textstyle\sum_{r \in C} \texttt{dist}(r,d)^2$$
$$\texttt{from\_up}(r,d) = \begin{cases} 0 & \text{if } r \leq a_2 \\ r - a_2 & \text{if } r > a_2 \end{cases} \qquad \texttt{from\_up}(C,d) = \textstyle\sum_{r \in C} \texttt{from\_up}(r,d)^2$$
$$\texttt{from\_low}(r,d) = \begin{cases} 0 & \text{if } r < a_1 \\ r - a_1 & \text{if } r \geq a_1 \end{cases} \qquad \texttt{from\_low}(C,d) = \textstyle\sum_{r \in C} \texttt{from\_low}(r,d)^2$$

□



DOI: 10.1002/stvr

By combining inclusion relations and distance functions, it is possible to evaluate the conformance of the implementation with respect to the specification in different ways.

*Definition 9*
Let $I$ and $S$ be two IFSMs, $H$ be a finite multiset of timed executions of $I$, $\Phi = \{e \mid \exists\, t : (e,t) \in H\} \cap \mathtt{NTEvol}(S)$, $0 \leq \alpha \leq 1$, and $\beta \in \mathbb{R}_+$.

(a) $I$ $(H,\alpha)$-*timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{int}^{(H,\alpha)}\ S$,
(b) $I$ $(H,\alpha)$-*preferable timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{intp}^{(H,\alpha)}\ S$,
(c) $I$ $(H,\alpha)$-*fast timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S$,
(d) $I$ $(H,\beta)$-*global timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{intgb}^{(H,\beta)}\ S$,
(e) $I$ $(H,\beta)$-*up-timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{intup}^{(H,\beta)}\ S$,
(f) $I$ $(H,\beta)$-*low-timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{intlw}^{(H,\beta)}\ S$,
(g) $I$ $(H,\alpha,\beta)$-*timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{int}^{(H,\alpha,\beta)}\ S$,
(h) $I$ $(H,\alpha,\beta)$-*preferable timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{intp}^{(H,\alpha,\beta)}\ S$, and
(i) $I$ $(H,\alpha,\beta)$-*fast timely conforms* to $S$, denoted by $I\ \mathtt{conf}_{intf}^{(H,\alpha,\beta)}\ S$,

respectively, if $I\ \mathtt{conf}_{nt}\ S$ and for all $e \in \Phi$ and all time intervals $d \in \mathcal{I}_{\mathbb{R}_+}$, if $(e,d) \in \mathtt{TEvol}(S)$ then, respectively,

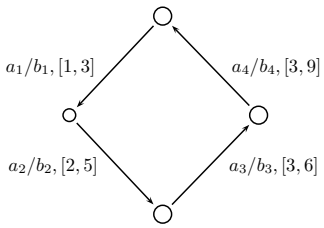
(a) $\mathtt{Obs\_Time}_H(e) \subseteq_\alpha d$
(b) $\mathtt{Obs\_Time}_H(e) \preceq_\alpha d$
(c) $\mathtt{Obs\_Time}_H(e) \ll_\alpha d$
(d) $\mathtt{dist}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$
(e) $\mathtt{from\_up}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$
(f) $\mathtt{from\_low}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$
(g) $\mathtt{Obs\_Time}_H(e) \subseteq_\alpha d \ \wedge\ \mathtt{dist}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$
(h) $\mathtt{Obs\_Time}_H(e) \preceq_\alpha d \ \wedge\ \mathtt{from\_low}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$
(i) $\mathtt{Obs\_Time}_H(e) \ll_\alpha d \ \wedge\ \mathtt{from\_low}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$

□

First, all of the previous relations establish that the implementation must conform to the specification in functional terms (that is, $I\ \mathtt{conf}_{nt}\ S$). In addition, a given proportion of observed execution time values must belong to the corresponding time intervals indicated by the specification ($(H,\alpha)$-*timely conforms*), optionally including the additional conditions that all time values over upper bounds are *forbidden* and that the interval of allowed time values is above/below lower bounds of intervals ($(H,\alpha)$-*preferable timely conforms* and $(H,\alpha)$-*fast timely conforms*, respectively). In fact, the latter additional condition allows to reinterpret the meaning of lower bounds of intervals, giving the opportunity to denote that time values must be *between* lower and upper bounds, or that they must be *under* lower bounds (so time values within the interval are only *partially* acceptable), respectively. Moreover, the relations $(H,\beta)$-*global timely conforms*, $(H,\beta)$-*up-timely conforms, and* $(H,\beta)$-*low-timely conforms* require that distances between observed execution time values and allowed time values do not exceed a established threshold. These first six relations are a straightforward application of time criteria given in Definition 8. In addition, proportion-based criteria and distances-based criteria can be combined, leading to new implementation relations. The last three relations of the previous definition consider both kinds of requests simultaneously, that is, relations demand conditions both over the proportion of observed time values being out of the interval and over the distances from these values to the corresponding interval. In fact, these three combined relations, that is $(H,\alpha,\beta)$-*timely conforms*, $(H,\alpha,\beta)$-*preferable timely conforms*, and $(H,\alpha,\beta)$-*fast timely conforms*, are a representative sample of the kind of combined relations that can be defined. Note that many other choices to combine proportions criteria and distances criteria do not provide valuable new relations. For instance, a relation combining $\preceq$ and from_up is equivalent to a relation based only on $\preceq$ (i.e., *preferable timely conforms*). Similarly, the combination of $\ll$ and from_up does not provide a new relation either.



DOI: 10.1002/stvr




| $e$ | $\texttt{Obs\_Time}_{H_1}(e)$ | $\texttt{Obs\_Time}_{H_2}(e)$ | interval |
|---|---|---|---|
| $e_1$ | $\{1.9, 2.3, 2.6, 2.8, 3\}$ | $\{1.2, 1.2, 1.3, 1.3, 1.4\}$ | $[1, 3]$ |
| $e_2$ | $\{2.9, 3.6, 3.8, 3.9, 3.9, 4.1\}$ | $\{3.1, 3.2, 3.2, 3.2, 3.6, 3.8\}$ | $[3, 8]$ |
| $e_3$ | $\{5.6, 5.6, 6.2, 6.5, 6.5, 6.8, 8.3, 8.5, 10.1, 10.1\}$ | $\{6.4, 6.4, 6.4, 6.5, 6.5, 6.7\}$ | $[6, 14]$ |
| $e_4$ | $\{7.4, 7.9, 9.3, 10.9, 10.9, 11.3, 11.6, 13.0, 13.2\}$ | $\{9.2, 9.2, 9.2, 9.2, 9.2, 9.3, 9.3, 9.3, 9.3\}$ | $[9, 23]$ |

where $e_1 = (a_1/b_1), e_2 = (a_1/b_1, a_2/b_2), e_3 = (a_1/b_1, a_2/b_2, a_3/b_3)$ and $e_4 = (a_1/b_1, a_2/b_2, a_3/b_3, a_4/b_4)$.

Figure 3. Examples of implementation relations.

The following two examples show and compare the features of the different implementation relations given in Definition 9. The first example is more *numeric* and it is useful to illustrate the different computations associated with the implementation relations. The second example uses several variants of a real system to intuitively illustrate which implementation relation should be used in each specific situation.

*Example 6*

Consider the specification depicted in Figure 3 (its initial state is the one placed on the top of the graph). Two multisets of timed executions, $H_1$ and $H_2$, have been collected from a system under test. The table in Figure 3, bottom, shows the projections of these multisets for each sequence of actions as well as the time interval that the specification indicates for the corresponding sequence.

All the implementation relations require some conditions with respect to the ratio of observed time values outside/inside the time intervals and the total number of values. The system $(H_1, \alpha)$-*timely conforms* and $(H_1, \alpha)$-*preferable timely conforms* to the specification for certain values of $\alpha$. On the one hand, there are no time values greater than the upper bounds. On the other hand, the number of values registered during the execution of the traces $e_1, e_2, e_3$, and $e_4$ that are smaller than the lower bound of the intervals is small: 0 ,1, 2, and 2, respectively. The greatest ratio of time values out of the bounds of the associated interval corresponds to the trace $e_4$ and takes the value $\frac{2}{9} = 0.22$. Therefore, the conditions "for all traces $e$ in the sample, $\texttt{Obs\_Time}_{H_1}(e) \subseteq_\alpha d$" and "for all traces $e$ in the sample, $\texttt{Obs\_Time}_{H_1}(e) \preceq_\alpha d$" are satisfied only if $\alpha$ takes a value less than or equal to $0.78$. Consider the set $H_2$. There are no values out of the bounds of the intervals. Therefore, the conditions hold for all levels of tolerance, that is, for all values of $\alpha$. Regarding the relation $(H_2, \alpha)$-*fast timely conforms*, the system only fulfills the requirement "for all traces $e$ in the sample $\texttt{Obs\_Time}_{H_2}(e) \ll_\alpha d$" when $\alpha$ is equal to $0$. This situation happens because all the timed execution values are between the bounds of the corresponding interval for each trace and, therefore, the systems is never *fast*.

The relation $(H, \beta)$-*up-timely conforms* only establishes restrictions over the values greater than the upper bound of the interval. Taking into account that both $H_1$ and $H_2$ contain values smaller than the upper bounds, the implementation $(H_1, \beta)$-*up-timely conforms* and $(H_2, \beta)$-*up-timely conforms* to the specification for all values of $\beta$. Regarding the $(H, \beta)$-*low-timely* conformance relation, the required condition is satisfied by the system with respect to the set of values $H_1$ if $\beta$ takes values greater than $53$, that corresponds to the maximum value of the function



DOI: 10.1002/stvr

$\texttt{from\_low}(\texttt{Obs\_Time}_{H_1}(e), d)$. In the case of $H_2$, it is enough that $\beta$ takes values greater than $1.47$ because all the time execution values are very close to the lower bounds of the intervals.

Finally, taking into account the previous requirements, if the parameters $\alpha$ and $\beta$ take appropriate values, the system $(H_i, \alpha, \beta)$-*timely, preferable timely,* and *fast timely conforms* to the specification, for $i \in \{1, 2\}$. □

The following simple example illustrates why it is interesting to have several implementation relations. In particular, given different variants of a system, it is important to chose the most appropriate implementation relation to capture, in a more suitable way, the notion of *good* implementation.

*Example 7*
Suppose the testing of a video player application. Specifically, consider the functionality that iteratively decodes the next frame and sends it to the video device. The time required to process and send the next frame should approximately match the time during which each frame is displayed on the screen, that is, ideally, this time should be within a given interval $[t_1, t_2]$. If the next frame is processed at time $t > t_2$, then there are two choices: Either delaying the display (*discontinuous-speed* reproduction) or refusing to display it when time $t_2$ is reached, thus keeping the previous frame (*skipped* reproduction). If the next frame is processed at time $t < t_1$, then it could be immediately displayed (discontinuous-speed reproduction again), it could be discarded (again, skipped reproduction), or it could be stored so that it is sent to the video device later (this requires that the player is provided with enough memory to keep early images until it displays them). Next, different scenarios are considered as well as the implementation relation that should be checked for each one.

Depending on the kind of errors detected, the chosen implementation relation should consider errors from a different point of view. In particular, if the consequences of early or late behaviors depend on *how* early or late they are produced, then distances should be considered. In this example, this is the case if early and late images are actually displayed when they are produced: Displaying a frame 0.2 seconds late is worse than displaying it 0.1 seconds late. However, if the consequences of early and late behaviors do not depend on the gap between the observed time and the expected time, then the *proportion* of wrong time values should be used. In this example, this is the case if early and late images are discarded: If after 0.3 seconds it is decided not to wait anymore and keep the previous frame, then it is irrelevant how late the delayed image will be produced.

Suppose that there is not enough memory available to store images produced too early for being displayed later, and both early and late images are just discarded. In this case, the tester should be concerned only about the *number* of discarded images. Therefore, the $(H, \alpha)$-*timely* relation is suitable. Furthermore, assume that discarding an image due to late processing is just unacceptable. In this case, a good choice is $(H, \alpha)$-*preferable timely*. Alternatively, suppose that there is enough memory available to store early images (i.e., early images can be displayed later at the expected time) and late images are discarded (i.e., late images produce skipped reproduction). Moreover, consider that producing an image *too* late is completely unacceptable. In this case, the $(H, \alpha)$-*fast timely* relation should be chosen. Regarding the previous *too late* limit, note that this relation, as well as $(H, \alpha, \beta)$-*fast timely*, considers both *late* and *too late* limits. This is done by assuming that the *late* limit is denoted by the *lower* bound of each interval, while the upper bound denotes the *too late* limit. This interpretation allows to denote that any time beyond the late limit is not good, but any time beyond the too late limit is unacceptable.

Assume that there is not enough memory to store early images and both early and late images are actually sent to the video device. In this situation, the tester should be concerned about *how early* and *how late* are early/late images. In this case, the $(H, \beta)$-*global timely* relation should be used. If there is enough memory to properly handle early images but late images are actually (lately) displayed then the tester should be concerned about how late are these images. Therefore, either the $(H, \beta)$-*up-timely* relation or the $(H, \beta)$-*low-timely* relation should be used, depending on how lower and upper bounds of the specification are used. In addition, a more flexible approach, where both early and late images may be displayed/discarded depending on run-time conditions, can be considered. For instance, memory limitations could bind the maximal number of early images that




DOI: 10.1002/stvr

can be stored to be displayed later on time and a given displaying policy could limit the number of images that are discarded due to late processing. In this case the $(H, \alpha, \beta)$-*timely* relation should be used because this relation accounts both the proportion of time values that are outside the interval and the distances from these values to the interval. There are other possibilities. If early images are discarded, images exceeding upper bounds of intervals are unacceptable, and displaying images later than the *lower* bounds of intervals is considered *bad*, then the $(H, \alpha, \beta)$-*preferable* relation should be chosen. However, if the memory allows the device to handle early images, late images can be either displayed or discarded, and too late images are unacceptable, then the $(H, \alpha, \beta)$-*fast timely* relation should be used.

In conclusion, deciding which implementation relation should be checked strongly depends on the kind of system and on the kind of behaviors that are considered acceptable in each case. □

Note that having the previously defined relations parameterized by the set $H$ is somehow similar to consider the, widely used, *fairness assumption* in formal testing: If a system $I$ is tested *enough*, it can be assumed that all the possible paths of the tested system were exercised. The fairness assumption essentially corresponds to have $H = \mathtt{TEvol}(I)$ while the framework presented in this paper considers that an implementation is correct up to a certain submultiset of $\mathtt{TEvol}(I)$. Note that the fairness assumption is related to the more general notion of *fairness* in systems where it is assumed that if a possibility is available enough times then it will be eventually chosen (see the book [41] for a good discussion on fairness and different possibilities to define appropriate notions).

Some properties of the proposed implementation relations are identified next. Implications identified in part (c) of the following proposition are graphically depicted in Figure 4.

*Proposition 1* (a) *(reflexivity)* Let $M$ be an IFSM, $H$ be a multiset of timed executions of $M$, $0 \leq \alpha \leq 1$, and $\beta \in \mathbf{R}_+$. Let $\mathtt{conf}^{(H,\alpha,\beta)}$ be any of the implementation relations given in (a), (b), (d), (e), or (g) of Definition 9 (ignore $\alpha$ or $\beta$ if they do not apply in that case). Then, $M\ \mathtt{conf}^{(H,\alpha,\beta)}\ M$.

(b) *(transitivity)* Let $A, B, C$ be input-enabled IFSMs and $\mathtt{conf}^{(H,\alpha,\beta)}$ be any of the implementation relations given in Definition 9. If $A\ \mathtt{conf}^{(H,\alpha,\beta)}\ B$ and $B\ \mathtt{conf}^{(H,\alpha,\beta)}\ C$ then $A\ \mathtt{conf}^{(H,\alpha,\beta)}\ C$.

(c) *(implications)* Let $A, B$ be IFSMs.

(c.1) If $A\ \mathtt{conf}_{intp}^{(H,\alpha)}\ B$ then $A\ \mathtt{conf}_{int}^{(H,\alpha)}\ B$,

(c.2) If $A\ \mathtt{conf}_{intlw}^{(H,\beta)}\ B$ then $A\ \mathtt{conf}_{intup}^{(H,\beta)}\ B$,

(c.3) If $A\ \mathtt{conf}_{int}^{(H,\alpha,\beta)}\ B$ then $A\ \mathtt{conf}_{int}^{(H,\alpha)}\ B$ and $A\ \mathtt{conf}_{intgb}^{(H,\beta)}\ B$,

(c.4) If $A\ \mathtt{conf}_{intp}^{(H,\alpha,\beta)}\ B$ then $A\ \mathtt{conf}_{intp}^{(H,\alpha)}\ B$, $A\ \mathtt{conf}_{intlw}^{(H,\beta)}\ B$, $A\ \mathtt{conf}_{int}^{(H,\alpha)}\ B$, and $A\ \mathtt{conf}_{intup}^{(H,\beta)}\ B$,

(c.5) If $A\ \mathtt{conf}_{intf}^{(H,\alpha,\beta)}\ B$ then $A\ \mathtt{conf}_{intf}^{(H,\alpha)}\ B$, $A\ \mathtt{conf}_{intlw}^{(H,\beta)}\ B$, and $A\ \mathtt{conf}_{intup}^{(H,\beta)}\ B$.

*Proof*

In order to prove (a), note that the relations $\mathtt{conf}^{(H,\alpha,\beta)}$ impose both non-temporal and temporal constraints. The non-temporal requirement of $\mathtt{conf}^{(H,\alpha,\beta)}$ is given by $\mathtt{conf}_{nt}$. By Definition 5 it is straightforward to check that $\mathtt{conf}_{nt}$ is reflexive. Regarding temporal constraints, note that all relations given in (a), (b), (d), (e), (g) of Definition 9 accept all execution time values that are both over the lower bound *and* under the upper bound of the IFSM at the right hand side of $\mathtt{conf}^{(H,\alpha,\beta)}$. Since $H$ is a multiset of timed executions of the IFSM at the left hand side of $\mathtt{conf}^{(H,\alpha,\beta)}$, all execution time values are within the corresponding intervals of the IFSM at the left side. Thus, the temporal requirement of $\mathtt{conf}^{(H,\alpha,\beta)}$ holds and $M\ \mathtt{conf}^{(H,\alpha,\beta)}\ M$.

In order to prove (b), assume that $A\ \mathtt{conf}^{(H,\alpha,\beta)}\ B$ and $B\ \mathtt{conf}^{(H,\alpha,\beta)}\ C$. Then, in particular, $A\ \mathtt{conf}_{nt}\ B$ and $B\ \mathtt{conf}_{nt}\ C$. Since $A$ and $B$ are input-enabled, the $\mathtt{conf}_{nt}$ relation reduces to trace containment. Therefore, $\mathtt{NTEvol}(A) \subseteq \mathtt{NTEvol}(C)$, concluding $A\ \mathtt{conf}_{nt}\ C$. Regarding temporal constraints imposed by $\mathtt{conf}^{(H,\alpha,\beta)}$, note that $A\ \mathtt{conf}^{(H,\alpha,\beta)}\ B$ and $B\ \mathtt{conf}^{(H,\alpha,\beta)}\ C$ imply that $H$ is a multiset of timed executions of *both* $A$ and $B$. The temporal requirement of $\mathtt{conf}^{(H,\alpha,\beta)}$ is assessed by comparing the multiset $H$ of timed executions of the IFSM at the left



DOI: 10.1002/stvr

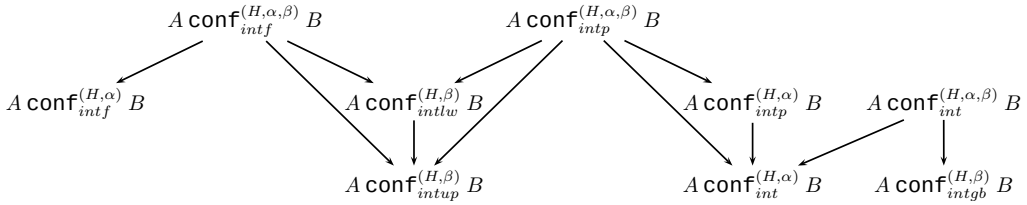



Figure 4. Implications between relations.

hand side with the IFSM at the right hand side. Thus, if $B\ \mathtt{conf}^{(H,\alpha,\beta)}\ C$ then the temporal requirement of $A\ \mathtt{conf}^{(H,\alpha,\beta)}\ C$ holds as well, since the same multiset $H$ of timed executions is considered. Therefore, $A\ \mathtt{conf}^{(H,\alpha,\beta)}\ C$.

The five subcases of (c) are proved next. First, (c.1) is considered. If $A\ \mathtt{conf}^{(H,\alpha)}_{intp}\ B$ then $A\ \mathtt{conf}_{nt}\ B$ and the execution time values collected from $A$ fulfill the following conditions:

- They are never greater than the upper bounds imposed by $B$ and
- the proportion of execution time values smaller than lower bounds imposed by $B$ is smaller than $1-\alpha$.

These two conditions imply, in particular, that the proportion of execution time values out of the intervals imposed by $B$ is below $1-\alpha$. Since $A\ \mathtt{conf}_{nt}\ B$, it can be concluded that $A\ \mathtt{conf}^{(H,\alpha)}_{int}\ B$. Next, (c.2) is proved. For a given time interval $[a_1, a_2]$ imposed by $B$ for some trace, let $Q$ be the set of all execution time values of $H$ for the corresponding trace that are greater than $a_1$. Besides, let $Q' \subseteq Q$ be the set of all execution time values of $H$ that are greater than $a_2$. It is trivial to check that the addition of squares of distances from time values in $Q$ to $a_1$ is *lower than or equal to* the addition of squares of distances from time values in $Q' \subseteq Q$ to $a_2$ since $Q'$ is contained in $Q$, $a_2 \geq a_1$, and $Q'$ contains only values greater than $a_2$. Given the fact that $A\ \mathtt{conf}_{nt}\ B$, it can be concluded that $A\ \mathtt{conf}^{(H,\beta)}_{intup}\ B$. Regarding the three implications given in (c.3), (c.4), and (c.5), note that the relation assumed in the premise of each of these three cases imposes *two* conditions over temporal samples. In particular, in (c.3) the premise $A\ \mathtt{conf}^{(H,\alpha,\beta)}_{int}\ B$ requires, in addition to $I\ \mathtt{conf}_{nt}\ S$, that for all $e \in \Phi$ and all time intervals $d \in \mathcal{I}_{\mathbb{R}_+}$, if $(e,d) \in \mathtt{TEvol}(S)$ then two conditions must hold: $\mathtt{Obs\_Time}_H(e) \subseteq_\alpha d$ and $\mathtt{dist}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$. Thus, $A\ \mathtt{conf}^{(H,\alpha,\beta)}_{int}\ B$ trivially implies the relation imposing that requirement *only* over $\mathtt{Obs\_Time}_H(e) \subseteq_\alpha d$ (that is, $\mathtt{conf}^{(H,\alpha)}_{int}$) and the relation imposing that requirement *only* over $\mathtt{dist}(\mathtt{Obs\_Time}_H(e), d) \leq \beta$ (that is, $\mathtt{conf}^{(H,\beta)}_{intgb}$). For a similar reason, in (c.4) the relation $\mathtt{conf}^{(H,\alpha,\beta)}_{intp}$ implies $\mathtt{conf}^{(H,\alpha)}_{intp}$ and $\mathtt{conf}^{(H,\beta)}_{intlw}$. The other two relations implied by $\mathtt{conf}^{(H,\alpha,\beta)}_{intp}$ in (c.4), that is $\mathtt{conf}^{(H,\alpha)}_{int}$ and $\mathtt{conf}^{(H,\beta)}_{intup}$, are a consequence of (c.1) and having $\mathtt{conf}^{(H,\alpha)}_{intp}$, and a consequence of (c.2) and having $\mathtt{conf}^{(H,\beta)}_{intlw}$, respectively. Similarly, in (c.5) the relation $\mathtt{conf}^{(H,\alpha,\beta)}_{intf}$ contains the temporal conditions required by $\mathtt{conf}^{(H,\alpha)}_{intf}$ and $\mathtt{conf}^{(H,\beta)}_{intlw}$, while $\mathtt{conf}^{(H,\beta)}_{intup}$ is a consequence of $\mathtt{conf}^{(H,\beta)}_{intlw}$ by (c.2). □

Note that the implementation relations given in (c), (f), (h), and (i) of Definition 9 are not reflexive. This is so because relations based either on $\ll_\alpha$ or on from_low do not necessarily accept all execution time values that are within the intervals given by the IFSM placed at the right side of the relation. Intuitively, this result means that, in these relations, a good implementation must be faster than the specification, and therefore, the specification itself cannot be a good implementation. Besides, it is worth to point out that the only implications between pairs of relations given in Definition 9 are those identified in the previous result.



DOI: 10.1002/stvr

The next result identifies some special conditions that make the previous relations collapse. The idea is that sometimes specifications follow a predefined pattern regarding temporal behaviors. For example, if it is assumed that taking *low* time values is always allowed, then specifications could use only time intervals of the form $[0, a_2]$. This is the case if, for example, time information is associated with the time required to compute a value (getting faster the result of a computation is generally good). Alternatively, taking only intervals of the form $[a_1, \infty]$ implicitly denotes that *high* time values are not bad. This would be suitable if time information denotes the time consumed before some kind of undesired behavior is produced. Other specifications could denote that there is a *single* ideal time for each event (i.e., only intervals $[a_1, a_1]$ are used) and any other time is not appropriate. In these special circumstances, some of the implementation relations given in Definition 9 collapse, that is, they denote the same requirements.

*Proposition 2*
Let $I, S$ be two IFSMs and consider the following IFSMs:

1. $S'$ coincides with $S$ but replacing its set of transtions by the set $Tr' = \{(s, s', i, o, [0, a_2]) | (s, s', i, o, [a_1, a_2]) \in Tr\}$.
2. $S''$ coincides with $S$ but replacing its set of transtions by the set $Tr'' = \{(s, s', i, o, [a_2, a_2]) | (s, s', i, o, [a_1, a_2]) \in Tr\}$.
3. $S'''$ coincides with $S$ but replacing its set of transtions by the set $Tr''' = \{(s, s', i, o, [a_1, \infty]) | (s, s', i, o, [a_1, a_2]) \in Tr\}$.
4. $S''''$ coincides with $S$ but replacing its set of transtions by the set $Tr'''' = \{(s, s', i, o, [a_2, \infty]) | (s, s', i, o, [a_1, a_2]) \in Tr\}$.

The following properties hold:

(a) $I\ \mathtt{conf}_{int}^{(H,\alpha)}\ S'''$ iff $I\ \mathtt{conf}_{intp}^{(H,\alpha)}\ S'''$
(b) $I\ \mathtt{conf}_{int}^{(H,\alpha)}\ S'$ iff $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S''''$
(c) $I\ \mathtt{conf}_{intp}^{(H,\alpha)}\ S'$ iff $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S''$
(d) $I\ \mathtt{conf}_{intp}^{(H,\alpha)}\ S''$ iff $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S'$
(e) $I\ \mathtt{conf}_{intgb}^{(H,\beta)}\ S'$ iff $I\ \mathtt{conf}_{intup}^{(H,\beta)}\ S'$
(f) $I\ \mathtt{conf}_{intgb}^{(H,\beta)}\ S'$ iff $I\ \mathtt{conf}_{intlw}^{(H,\beta)}\ S''''$
(g) $I\ \mathtt{conf}_{intlw}^{(H,\beta)}\ S''$ iff $I\ \mathtt{conf}_{intup}^{(H,\beta)}\ S$
(h) $I\ \mathtt{conf}_{intup}^{(H,\beta)}\ S'''$ iff $I\ \mathtt{conf}_{nt}\ S$
(i) $I\ \mathtt{conf}_{int}^{(H,\alpha,\beta)}\ S'$ iff $I\ \mathtt{conf}_{intf}^{(H,\alpha,\beta)}\ S''''$

*Proof*
Only the proof of (c) is given since the rest of the results can be proved by using similar arguments.

Consider the implication from left to right. Let $\Phi = \{e \mid \exists\, t : (e, t) \in H\} \cap \mathtt{NTEvol}(S')$ and suppose $I\ \mathtt{conf}_{intp}^{(H,\alpha)}\ S'$. Then, $I\ \mathtt{conf}_{nt}\ S'$ and for all $e \in \Phi$ and $d \in \mathcal{I}_{\mathbb{R}_+}$, $(e, d) \in \mathtt{TEvol}(S')$ implies $\mathtt{Obs\_Time}_H(e) \preceq_\alpha d$. That is, for all $(e, [a_1, a_2]) \in \mathtt{TEvol}(S')$, $\|\{r \mid r \in \mathtt{Obs\_Time}_H(e) \wedge r > a_2\}\| = 0$ and $\frac{\|\{r \mid r \in \mathtt{Obs\_Time}_H(e) \wedge r < a_1\}\|}{\|\mathtt{Obs\_Time}_H(e)\|} \leq 1 - \alpha$. Next, it is necessary to check that the conditions required to fulfill $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S''$ actually hold. The first requirement, $I\ \mathtt{conf}_{nt}\ S''$, trivially holds because $I\ \mathtt{conf}_{nt}\ S'$ and time intervals do not affect $\mathtt{conf}_{nt}$. For this reason, $\Phi = \{e \mid \exists\, t : (e, t) \in H\} \cap \mathtt{NTEvol}(S'')$. The second requirement of $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S''$ is that for all $e \in \Phi$ and all $d \in \mathcal{I}_{\mathbb{R}_+}$, $(e, d) \in \mathtt{TEvol}(S'')$ implies $\mathtt{Obs\_Time}_H(e) \ll_\alpha d$. That is, it is necessary to prove that for all $(e, [a_1, a_2]) \in \mathtt{TEvol}(S'')$, $\|\{r \mid r \in \mathtt{Obs\_Time}_H(e) \wedge r > a_2\}\| = 0$ and $\frac{\|\{r \mid r \in \mathtt{Obs\_Time}_H(e) \wedge a_1 < r \leq a_2\}\|}{\|\mathtt{Obs\_Time}_H(e)\|} \leq 1 - \alpha$. By the construction of $S''$ from $S$, for all $(e, [a_1, a_2]) \in \mathtt{TEvol}(S'')$, $a_1 = a_2$. Moreover, by the construction of $S'$ from $S$, for all $(e, [a_2, a_2]) \in \mathtt{TEvol}(S'')$, $(e, [0, a_2]) \in \mathtt{TEvol}(S')$ holds and by the assumption of $I\ \mathtt{conf}_{intp}^{(H,\alpha)}$ $S'$, $\|\{r \mid r \in \mathtt{Obs\_Time}_H(e) \wedge r > a_2\}\| = 0$, which is the first condition required for all $(e, [a_2, a_2]) \in \mathtt{TEvol}(S'')$. The second requirement is in fact $\frac{\|\{r \mid r \in \mathtt{Obs\_Time}_H(e) \wedge a_2 < r \leq a_2\}\|}{\|\mathtt{Obs\_Time}_H(e)\|} \leq$



DOI: 10.1002/stvr

$1-\alpha$, which trivially holds because the interval $a_2 < r \leq a_2$ is empty. Therefore, $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S''$ holds.

In order to prove the right to left implication, note that $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S''$ implies that $I\ \mathtt{conf}_{nt}\ S''$ and for all $(e,[a_2,a_2]) \in \mathtt{TEvol}(S'')$, $\|\{r \mid r \in \mathtt{Obs\_Time}_H(e)\ \wedge\ r > a_2\}\| = 0$ and the (trivial) condition $\frac{\|\{r \mid r \in \mathtt{Obs\_Time}_H(e) \wedge a_2 < r \leq a_2\}\|}{\|\mathtt{Obs\_Time}_H(e)\|} \leq 1-\alpha$. So, by using similar arguments as before, $I\ \mathtt{conf}_{nt}\ S'$ and for all $(e,[0,a_2]) \in \mathtt{TEvol}(S'')$, $\|\{r \mid r \in \mathtt{Obs\_Time}_H(e)\ \wedge\ r > a_2\}\| = 0$ and the (trivial) condition $\frac{\|\{r \mid r \in \mathtt{Obs\_Time}_H(e) \wedge r < 0\}\|}{\|\mathtt{Obs\_Time}_H(e)\|} \leq 1-\alpha$. Thus, $I\ \mathtt{conf}_{intp}^{(H,\alpha)}\ S'$. □

## 4. DEFINITION AND APPLICATION OF TESTS

A test represents a sequence of inputs applied to the implementation under test. After applying each input, it is necessary to check whether the received output is expected 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. In order to test an implementation with input and output sets $I$ and $O$, respectively, tests are deterministic acyclic $I/O$ labeled transition systems (i.e. trees) with a strict alternation between an input action and the set of output actions. An output action is followed either by a leaf (indicating either failure or successful termination) or by an input action. Leaves are labeled either by *pass* or by *fail*. In the first case, a *time stamp* is also placed in the leaf. The time stamp will be a time interval. The idea is that the time that the implementation takes to reach that point will be compared with the time stamp.

*Definition 10*
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, with $s_0 \in S_I$. For all input states $s \in S_I$ there exists a unique outgoing transition $(s,a,s') \in Tr$. For this transition, $a \in I$ and $s' \in S_O$.
- $S_O$ is the set of *output* states. For all output states $s \in S_O$ and 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. These states are called *terminal*. That is, for all states $s \in S_F \cup S_P$, there do not exist $a \in I \cup O$ and $s' \in S$ such that $(s,a,s') \in Tr$.

Finally, $C_T : S_P \longrightarrow \mathcal{I}_{\mathbb{R}_+}$ is a function associating time stamps, that is, a time interval, with passing states.

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

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

From now on, the reader must assume that tests refer only to valid tests. Figure 5 shows three valid tests that can be applied to the running example. For the sake of clarity, in this graphical representations a branch labeled by `else` represents a separate branch for each output different from the one appearing to the right of the corresponding else-branch.

Next, the application of a test to an implementation is defined. A test suite $\mathcal{T}$ is *passed* if, for all tests, the terminal states reached by the composition of the implementation and the test belong to the set of *passing* states. Note that since implementations are input-enabled, the testing process will conclude only when the test reaches either a fail or a success state.



DOI: 10.1002/stvr

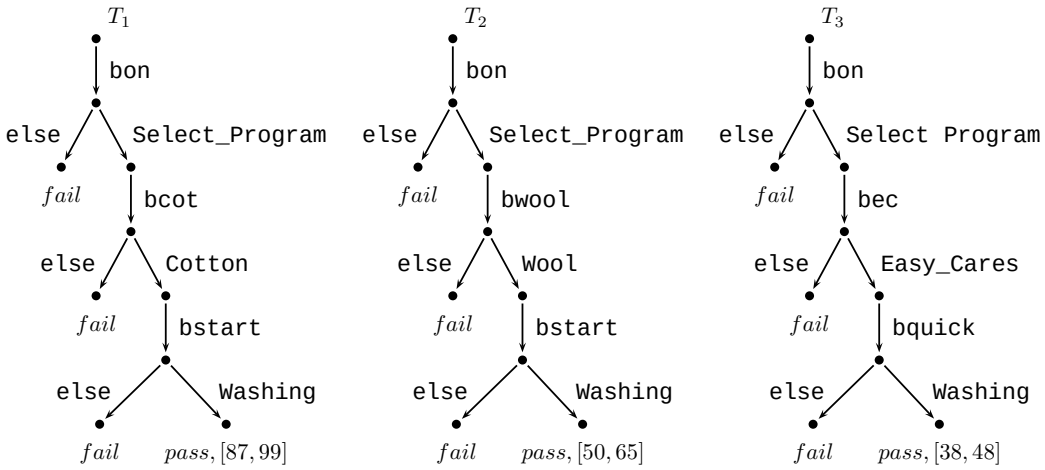



Figure 5. Three different tests.

*Definition 11*
Let $I$ be an IFSM, $T$ be a test, and $s$ be a state of $T$. $I \parallel T \stackrel{e}{\Longrightarrow} s$ denotes that $T \stackrel{e}{\Longrightarrow} s$ and $e \in \mathtt{NTEvol}(I)$.

$I$ *passes* the test suite $\mathcal{T}$, denoted by $\mathtt{pass}(I, \mathcal{T})$, if for all tests $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C_T) \in \mathcal{T}$ and $e \in \mathtt{NTEvol}(I)$ there does not exist $s \in S_F$ such that $I \parallel T \stackrel{e}{\Longrightarrow} s$. □

*Example 8*
Consider the running example presented in Figure 1 and the tests given in Figure 5. The system passes the test suite $\{T_1, T_2\}$ since there does not exist a computation that leads to a fail state. On the contrary, if the test suite $\{T_3\}$ is applied then a failing computation appears since, for example, the system and the test can perform the sequence

`bon,Select_Program,bec,Easy_Cares,bquick`

but the message displayed by the system after the button `bquick` is pressed, that is, `Quick_Wash`, leads the test to a fail state. □

The previous definition of passing tests does not take into account the time values that will be collected during the application of tests. The set of *observed timed executions* will be checked against time stamps. In fact, it is necessary to obtain a set of test executions associated to each evolution in order to evaluate if they match, in a certain sense, the time interval associated to the corresponding state of the test. In order to increase the reliability degree, the method presented in this paper does not follow the classical approach where passing a test suite is defined according only to the results for each test. In this method all the observations will be put together so that more samples for each evolution will be available. 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.

*Definition 12*
Let $I$ be an IFSM, $T$ be a test, and $s$ be a state of $T$. $I \parallel T \stackrel{e}{\Longrightarrow}_t s$ denotes that $T \stackrel{e}{\Longrightarrow} s$ and that there exists $d \in \mathcal{I}_{\mathbb{R}_+}$ such that $(e, d) \in \mathtt{TEvol}(I)$ and $t \in d$. In this case, $(e, t)$ is called a *test execution* of $I$ and $T$.

Let $I$ be an IFSM and $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a test suite. Let $H_1, \ldots, H_n$ be finite multisets of test executions of $I$ and $T_1, \ldots, T_n$, respectively. Let $H = \bigcup_{i=1}^{n} H_i$, $\Phi = \{e \mid \exists\, t : (e, t) \in H\}$, $\beta \in \mathbf{R}_+$, and $0 \leq \alpha \leq 1$. Then,



DOI: 10.1002/stvr

(a) $I$ $(H,\alpha)$-*timely passes* the test suite $\mathcal{T}$,
(b) $I$ $(H,\alpha)$-*preferable passes* the test suite $\mathcal{T}$,
(c) $I$ $(H,\alpha)$-*fast passes* the test suite $\mathcal{T}$,
(d) $I$ $(H,\beta)$-*global timely passes* the test suite $\mathcal{T}$,
(e) $I$ $(H,\beta)$-*up-timely passes* the test suite $\mathcal{T}$,
(f) $I$ $(H,\beta)$-*low-timely passes* the test suite $\mathcal{T}$,
(g) $I$ $(H,\alpha,\beta)$-*timely passes* the test suite $\mathcal{T}$,
(h) $I$ $(H,\alpha,\beta)$-*preferable passes* the test suite $\mathcal{T}$, and
(i) $I$ $(H,\alpha,\beta)$-*fast passes* the test suite $\mathcal{T}$

respectively, if $\texttt{pass}(I,\mathcal{T})$ and for all $e \in \Phi$ and all $T \in \mathcal{T}$, $I \parallel T \stackrel{e}{\Longrightarrow} s$ implies, respectively,

(a) $\texttt{Obs\_Time}_H(e) \subseteq_\alpha C_T(s)$
(b) $\texttt{Obs\_Time}_H(e) \preceq_\alpha C_T(s)$
(c) $\texttt{Obs\_Time}_H(e) \ll_\alpha C_T(s)$
(d) $\texttt{dist}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$
(e) $\texttt{from\_up}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$
(f) $\texttt{from\_low}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$
(g) $\texttt{Obs\_Time}_H(e) \subseteq_\alpha C_T(s) \ \wedge\ \texttt{dist}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$
(h) $\texttt{Obs\_Time}_H(e) \preceq_\alpha C_T(s) \ \wedge\ \texttt{from\_low}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$
(i) $\texttt{Obs\_Time}_H(e) \ll_\alpha C_T(s) \ \wedge\ \texttt{from\_low}(\texttt{Obs\_Time}_H(e), C_T(s)) \leq \beta$

□

Note that an observed timed execution does not contain the time interval associated with performing the evolution (that is, the addition of all the intervals corresponding to each transition of the implementation) but the addition of all the time values that elapsed between the reception of each input and the performance of its associated output.

Intuitively, an implementation passes a test if there does not exist an evolution leading to a fail state and certain conditions on observed values hold. The set $H$ corresponds to the observations of the (several) applications of the tests belonging to the test suite to the implementation under test. Thus, it must be decided whether, for each evolution $e$, the observed time values (that is, $\texttt{Obs\_Time}_H(e)$) *match* the definition of the time intervals appearing in the successful state of the tests corresponding to the execution of that evolution (that is, $C_T(s)$).

*Example 9*
Consider again the running example presented in Figure 1 and the tests given in Figure 5. The system passes the test suite $\{T_1\}$ for the $(H,\alpha)$-*timely*, $(H,\alpha)$-*preferable*, $(H,\beta)$-*global timely*, $(H,\beta)$-*up-timely*, and $(H,\alpha,\beta)$-*timely* notions, for any value of $\alpha$ and $\beta$. This is so because these notions of passing a test suite only impose restrictions over the time values observed out of the interval associated to the pass state. In this case, all the values collected from the system, that is, $\{\!\{87,88,88,89\}\!\}$, are included in the corresponding interval $[87,99]$. The notions $(H,\alpha)$-*fast* and $(H,\alpha,\beta)$-*fast* are based on the limitation of the number of time values within the interval given by the relation $\ll_\alpha$. The system only fulfills the requirement if the value of $\alpha$ is equal to zero, taking into account that $\texttt{Obs\_Time}_H(e) \ll_\alpha [87,99]$ only holds for this value. In addition, the system does not pass the test suite $\{T_1\}$ for the notions $(H,\beta)$-*up-timely*, $(H,\alpha,\beta)$-*preferable*, and $(H,\alpha,\beta)$-*fast*, if the value of $\beta$ is greater than $\texttt{dist}(\texttt{Obs\_Time}_H(e),[87,99]) = 6$.

The system passes the test suite $\{T_2\}$ according to the different notions depending on the tolerance degree established and the error degree of wrong values accepted. For example, assume that $\alpha = \beta = 0$ and consider the sample of observations included in the multiset $H$ described in Example 5. In this situation, the system $(H,\alpha)$-*timely*, $(H,\alpha)$-*preferable*, and $(H,\alpha)$-*fast passes* the test suite $\{T_2\}$. The same situation appears regarding the notions that considers the distance functions $\texttt{from\_low}(\,)$ and $\texttt{from\_up}(\,)$. However, the system fails according to the definition of $(H,\beta)$-*global timely* passing the test suite. The reason is that the value of $\texttt{dist}(\texttt{Obs\_Time}_H(e), C_T(s)) = \texttt{dist}(\{\!\{39,39,40,42\}\!\},[50,65]) = 406$ is greater than the value considered for $\beta$. □



DOI: 10.1002/stvr

## 5. DERIVATION OF TEST SUITES

This section introduces an algorithm to derive tests from a specification. The derived test suites are sound and complete with respect to the implementation relations introduced in Section 3. The basic idea underlying test derivation consists in traversing the specification in order to get all the possible evolutions in an appropriate way. First, some additional notation is needed.

*Definition 13*
Let $M = (S, I, O, Tr, s_{in})$ be an `IFSM`. For all $s \in S$ and $i \in I$, the set $\texttt{out}(s, i)$ is defined as

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

Given $s \in S, i \in I, o \in O$, and $d \in \mathcal{T}$, the function $\texttt{after}(s, i, o, d)$ is defined as

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

□

The function $\texttt{out}(s, i)$ computes the set of output actions associated with those transitions that can be executed from $s$ after receiving the input $i$. The function $\texttt{after}(s, i, o, d)$ computes the *situation* that is reached from a state $s$ after receiving the input $i$, producing the output $o$, when the duration of the testing process so far is given by $d$. A situation is a pair containing the reached state and the cumulated intervals since the system started its performance. Note that, due to the assumption that `IFSM`s are observable, $\texttt{after}(s, i, o, d)$ is uniquely determined. In addition, this function will be applied only when the side condition holds, that is, `error` will never be returned as result of applying `after`.

The algorithm to derive tests from a specification $M$ is given in Figure 6. This algorithm is non-deterministic and each application returns a single test. By considering the set of all tests that can be obtained by taking non-deterministic choices in all possible ways, a test suite is extracted from the specification. This test suite is called $\texttt{tests}(M)$.

The algorithm works as follows. 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 definition has not been completed yet (that is, its outgoing transitions have not been defined yet). A tuple $(s^M, d, s^T) \in S_{aux}$ indicates that the state $s^T$ of the test is not completed yet, the current state in the traversal of the specification is $s^M$, and the accumulated time interval in the specification from the initial state is given by $d$. The set $S_{aux}$ initially contains a tuple with the initial states (of both specification and test) and the initial situation of the process (that is, the time interval is $[0, 0]$). There are two possibilities for each tuple belonging to $S_{aux}$. It is important to remark that the second step can be applied only when the set $S_{aux}$ becomes a singleton. So, the derived tests correspond to valid tests as introduced in Definition 10. 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 labeled by this input. Then, the whole sets of outputs is considered. If the output is not expected by the specification (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, labeled by `else`, leading to a failing state. For the expected outputs (step 2.(f) of the algorithm), a transition with the corresponding output is created and the appropriate tuple is added to the set $S_{aux}$. Note that a (finite) test is constructed simply by considering a step where the second inductive case is not applied.

*Example 10*
The test $T_1$ given in Figure 5 is one of the tests that the algorithm may produce when applied to the system given in Figure 1. Initially, the input `bon` is chosen in the step 2.b of the algorithm and a transition labeled by this input is generated in the test. The considered input determines the expected output(s). In this case, only `Select_Program` can be displayed on the screen. Thus, a transition for the `Select_Program` output is created in the test (step 2.f of the algorithm). Moreover, another transition leading to a fail state is created for the rest of the outputs, represented



DOI: 10.1002/stvr

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

*Initialization:*

- $S' := \{s_0\}; Tran', S_I, S_O, S_F, S_P := \emptyset$.
- $S_{aux} := \{(s_{in}, [0,0], s_0)\}$.

*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:
   (a) $S_{aux} := S_{aux} - \{(s^M, d, s^T)\}$.
   (b) $S_P := S_P \cup \{s^T\}$; $C_T(s^T) := d$.
2. If $S_{aux} = \{(s^M, d, s^T)\}$ and $\exists\, i \in I : \mathtt{out}(s^M, i) \neq \emptyset$ then perform the following steps:
   (a) $S_{aux} := \emptyset$.
   (b) Choose $i$ such that $\mathtt{out}(s^M, i) \neq \emptyset$.
   (c) Consider a fresh state $s' \notin S'$ and let $S' := S' \cup \{s'\}$.
   (d) $S_I := S_I \cup \{s^T\}$; $S_O := S_O \cup \{s'\}$; $Tran' := Tran' \cup \{(s^T, i, s')\}$.
   (e) For all $o \notin \mathtt{out}(s^M, i)$ do
      - Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
      - $S_F := S_F \cup \{s''\}$; $Tran' := Tran' \cup \{(s', o, s'')\}$.
   (f) For all $o \in \mathtt{out}(s^M, i)$ do
      - Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
      - $Tran' := Tran' \cup \{(s', o, s'')\}$.
      - $(s_1^M, d') := \mathtt{after}(s^M, i, o, d)$.
      - $S_{aux} := S_{aux} \cup \{(s_1^M, d', s'')\}$.

Figure 6. Derivation of tests from an observable specification.

by the label `else`. After this, the input `bcot` is selected. The only output that the specification can perform is the display of the `Cotton` message; a new transition associated to this output is included in the test. For the rest of the outputs, the corresponding transition leading to a fail state is created. The specification moves to state `sc` and the input `bstart` is selected. The appropriate transitions are created and, finally, step 1 of the algorithm is applied in order to conclude the generation of this test. Only one pass state is created. The attached time interval represents the associated time values with the different possibilities to perform the complete input/output sequence. Note that the algorithm considers only the lapses, extracted from the specification, between reception of inputs and performance of outputs. □

The main result of the paper shows that passing the test suite $\mathtt{tests}(S)$ for a sample $H$ is equivalent to having the conformance of the implementation under test with respect to $S$, according to the relations given in Definition 9.

*Theorem 1*
Let $I$ and $S$ be `IFSM`s. Given a finite multiset $H$ of timed executions of $I$, $\beta \in \mathbf{R}_+$, and $0 \leq \alpha \leq 1$,

(a) $I\ \mathtt{conf}_{int}^{(H,\alpha)}\ S$ iff $I$ $(H, \alpha)$-*timely passes* $\mathtt{tests}(S)$.
(b) $I\ \mathtt{conf}_{intf}^{(H,\alpha)}\ S$ iff $I$ $(H, \alpha)$-*fast passes* $\mathtt{tests}(S)$.
(c) $I\ \mathtt{conf}_{intp}^{(H,\alpha)}\ S$ iff $I$ $(H, \alpha)$-*preferable passes* $\mathtt{tests}(S)$.
(d) $I\ \mathtt{conf}_{intgb}^{(H,\beta)}\ S$ iff $I$ $(H, \beta)$-*global timely passes* $\mathtt{tests}(S)$.
(e) $I\ \mathtt{conf}_{intup}^{(H,\beta)}\ S$ iff $I$ $(H, \beta)$-*up-timely passes* $\mathtt{tests}(S)$.



DOI: 10.1002/stvr

(f) $I\ \mathtt{conf}_{intlw}^{(H,\beta)}\ S$ iff $I$ $(H,\beta)$-*low-timely passes* $\mathtt{tests}(S)$.
(g) $I\ \mathtt{conf}_{int}^{(H,\alpha,\beta)}\ S$ iff $I$ $(H,\alpha,\beta)$-*timely passes* $\mathtt{tests}(S)$.
(h) $I\ \mathtt{conf}_{intf}^{(H,\alpha,\beta)}\ S$ iff $I$ $(H,\alpha,\beta)$-*fast passes* $\mathtt{tests}(S)$.
(i) $I\ \mathtt{conf}_{intp}^{(H,\alpha,\beta)}\ S$ iff $I$ $(H,\alpha,\beta)$-*preferable passes* $\mathtt{tests}(S)$.

*Proof*
Only the proof of (g) is given since the proofs of the rest of results are similar.

The right to left implication will be proved by counterpositive. Therefore, assume that $I\ \mathtt{conf}_{int}^{(H,\alpha,\beta)}\ S$ does not hold. There are two possibilities.

First, suppose that the previous relation does not hold because $I\ \mathtt{conf}_{nt}\ S$ does not hold. This means that there are two non-timed evolutions $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r)$ and $e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r)$, with $r \geq 1$, such that $e \in \mathtt{NTEvol}(S)$, $e' \in \mathtt{NTEvol}(I)$, and $e' \notin \mathtt{NTEvol}(S)$. It will be shown that if $e \in \mathtt{NTEvol}(S)$ then there exists $T = (S_T, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C_T) \in \mathtt{tests}(S)$ such that $T \stackrel{e}{\Longrightarrow} s^T$ and $s^T \in S_P$. This test is built by applying the algorithm given in Figure 6 and solving non-deterministic choices as follows:

**for** $1 \leq j \leq r$ **do**
  apply inductive case 2 for input $i_j$
  apply inductive case 1 for all $(s^M, d, s^T) \in S_{aux}$ obtained
    by processing any output different from $o_j$
**end**
apply inductive case 1 for the last $(s^M, d, s^T) \in S_{aux}$

This test $T$ is such that $T \stackrel{e'}{\Longrightarrow} u^T$, for some $u^T \in S_F$. This is because the output $o'_r$ is necessarily treated in step $(e)$ during the last application of the inductive case 2, since $e' \notin \mathtt{NTEvol}(S)$. Given the fact that $T \in \mathtt{tests}(S)$, $\mathtt{pass}(I, \mathtt{tests}(S))$ does not hold. Thus, $I$ $(H,\alpha,\beta)$-*timely passes* $\mathtt{tests}(S)$ does not hold.

Second, suppose that $I\ \mathtt{conf}_{int}^{(H,\alpha,\beta)}\ S$ does not hold because there exists a sequence $e \in \{e' \mid \exists\, t : (e', t) \in H\} \cap \mathtt{NTEvol}(S)$ and a time interval $d \in \mathcal{I}_{\mathbb{R}_+}$ such that $(e, d) \in \mathtt{TEvol}(S)$ and either $\mathtt{Obs\_Time}_H(e) \not\subseteq_\alpha d$ or $\mathtt{dist}(\mathtt{Obs\_Time}_H(e), d) > \beta$. Consider the test $T$ built before for the non-timed evolution $e$. Since $e \in \mathtt{NTEvol}(S)$, $s^T \in S_P$ for a state $s^T$ satisfying $T \stackrel{e}{\Longrightarrow} s^T$. Besides, since $H$ is a multiset of timed executions of $I$, for all sequences $e \in \{e' \mid \exists\, t : (e', t) \in H\} \cap \mathtt{NTEvol}(S)$, $e \in \mathtt{NTEvol}(I)$. Therefore, $I \parallel T \stackrel{e}{\Longrightarrow} s^T$. According to the construction of $T$, $C_T(s^T)$ denotes the interval of time values allowed by $S$ to execute $e$, that is, $C_T(s^T) = d$. Hence, if $\mathtt{Obs\_Time}_H(e) \not\subseteq_\alpha d$ then $\mathtt{Obs\_Time}_H(e) \not\subseteq_\alpha C_T(s^T)$ and thus $I$ $(H,\alpha,\beta)$-*timely passes* $\mathtt{tests}(S)$ does not hold. In addition, if $\mathtt{dist}(\mathtt{Obs\_Time}_H(e), d) > \beta$ then $\mathtt{dist}(\mathtt{Obs\_Time}_H(e), C_T(s^T)) > \beta$ and $I$ $(H,\alpha,\beta)$-*timely passes* $\mathtt{tests}(S)$ does not hold either.

The left to right implication will be also proved by counterpositive. Therefore, assume that $I$ $(H,\alpha,\beta)$-*timely passes* $\mathtt{tests}(S)$ does not hold.

First, suppose that $I$ $(H,\alpha,\beta)$-*timely passes* $\mathtt{tests}(S)$ does not hold because $\mathtt{pass}(I, \mathtt{tests}(S))$ does not hold. This means that there exists a test $T \in \mathtt{tests}(S)$, $e' = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o'_r)$, and $s^T$ such that $I \parallel T \stackrel{e'}{\Longrightarrow} s^T$ and $s^T \in S_F$. Then, $e' \in \mathtt{NTEvol}(I)$ and $T \stackrel{e'}{\Longrightarrow} s^T$. According to the test derivation algorithm, a branch of a test leads to a failure state only if the associated output cannot be produced in the specification. Hence, $e' \notin \mathtt{NTEvol}(S)$. Note that the algorithm creates a failure state only when the inductive case 2 is applied. Applying this case requires that $\mathtt{out}(S_M, i) \neq \emptyset$, that is, the specification must provide some output in response to the chosen input. Thus, there exists an output $o_r$ and a non-timed evolution $e = (i_1/o_1, \ldots, i_{r-1}/o_{r-1}, i_r/o_r)$ such that $e \in \mathtt{NTEvol}(S)$. Since $e' \in \mathtt{NTEvol}(I)$, $e' \notin \mathtt{NTEvol}(S)$, and $e \in \mathtt{NTEvol}(S)$, it can be concluded that $I \not\mathtt{conf}_{nt} S$. Thus, $I\ \mathtt{conf}_{int}^{(H,\alpha,\beta)}\ S$ does not hold.



DOI: 10.1002/stvr

Second, assume that $I$ $(H, \alpha, \beta)$-*timely passes* $\texttt{tests}(S)$ does not hold but $\texttt{pass}(I, \texttt{tests}(S))$ holds. In this case, there must exist a sequence $e \in \{e' \mid \exists\, t : (e', t) \in H\}$ and a test $T = (S_T, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C_T) \in \mathcal{T}$, with $I \parallel T \stackrel{e}{\Longrightarrow} s^T$, such that either $\texttt{Obs\_Time}_H(e) \not\subseteq_\alpha C_T(s^T)$ or $\texttt{dist}(\texttt{Obs\_Time}_H(e), C_T(s^T)) > \beta$. Note that, since $\texttt{pass}(I, \texttt{tests}(S))$ holds, $s^T \in S_P$. By the construction of $\texttt{tests}(S)$ according to the test derivation algorithm given in Figure 6, $C_T(s^T)$ denotes the interval of time values allowed by $S$ to execute $e$, that is, for some $d \in \mathcal{I}_{\mathbb{R}_+}$, $(e, d) \in \texttt{TEvol}(S)$ and $C_T(s^T) = d$. Thus, if $\texttt{Obs\_Time}_H(e) \not\subseteq_\alpha C_T(s^T)$ then $\texttt{Obs\_Time}_H(e) \not\subseteq_\alpha d$ and $I \not\texttt{conf}_{int}^{(H,\alpha,\beta)} S$. In addition, if $\texttt{dist}(\texttt{Obs\_Time}_H(e), C_T(s^T)) > \beta$ then $\texttt{dist}(\texttt{Obs\_Time}_H(e), d) > \beta$. Therefore, $I\ \texttt{conf}_{int}^{(H,\alpha,\beta)}\ S$ does not hold. □

## 6. RELATED WORK

This section briefly reviews some related work. The extensions of process algebras with time are somehow related to the proposal presented in this paper since process algebras usually rely on semantic models, either equivalence relations or (pre-)orders, to relate processes. The following proposals [15, 42, 43, 44, 45, 46, 18, 47] can be mentioned among many others. There is a main difference between the work on timed process algebras and the work reported in this paper. In the former, the considered semantic relation relates two processes generated from the same language, that is, both processes have the same nature and the *code* of both of them is visible. In contrast, the proposal presented in this paper relates two objects of different nature: One of them is visible (the specification) and the other one is not (the implementation). Even assuming that the implementation is written in the same language as the specification, it is not possible to access its textual description. This fact produces that implementation relations asymmetrically relate processes: They compare intervals of the specification with time values extracted from the implementation. Similarly, even though the objective is similar, applying tests to systems, this proposal is not very related to the classical de Nicola & Hennessy's testing theory [48, 49]. The latter is based on comparing two processes by analyzing their responses to *all* the tests of a given set of tests. In addition, processes are always visible, so that their *branching* can be fully observed. On the contrary, the approach presented in this paper consists in applying tests, derived from specifications, to implementations to determine whether the implementation is *somehow* correct with respect to the specification.

There are already several papers devoted to formal testing of timed systems (for example, the papers [21, 22, 23, 24, 25, 35, 50], to name a few). The interested reader is referred to the paper [28] where a comparison between different approaches is presented. Even though (formal) timed testing can be considered a mature area of research, only a few papers directly or indirectly consider the topic of *non-strict* deadlines, and this number is even more reduced when restricted to a testing framework. The work reported in the paper [51] presents a probabilistic formalism to approximate the idea of soft deadline. However, this approach is not very related to the one presented in this paper since, on the one hand, they are based on the work [48, 49], and, on the other hand, they use a probabilistic approach, based on the paper [52], to deal with soft deadlines. As mentioned in the introduction of the paper, stochastic models allow to partially simulate soft deadlines. In this line, there are two proposals to test stochastic systems [53, 54]. but they are also inspired on the work [48, 49].

Although not directly related to this framework, it is worth to mention the topic of testing non-strict requirements over some kind of numerical values (not specifically time) as presented in the paper [55]. A *distance* between two systems is calculated by measuring how far the numerical values associated to actions in one of them are from the values associated to the same actions in the other one. In particular, the distance between two *traces* composed of the same actions is given by the maximum difference between values associated to the same steps in both sequences. Based on this idea, the distance between two *systems* is the highest distance between traces of both systems, where each trace of one system is compared to the *nearest* trace of the other. In the paper [56], this distances approach is considered in a framework where numerical values explicitly represent time,



DOI: 10.1002/stvr

although the goal is not to develop a black-box testing methodology to check the correctness of an implementation under test, but to directly compare/study models in a white-box fashion. This is also the goal of the work reported in the paper [57], where semantic relations, such as traces inclusion, traces equivalence, simulation, and bisimulation, are developed for a similar (but not specifically timed) quantitative framework. In the paper [58], several functions associating each trace to a single numerical value, depending on the values associated to actions in the trace, are considered and studied (e.g. maximum, sum, different kinds of limits for infinite traces, etc).

In contrast with these approaches to non-strict requirements, which are based on the previously described notion of distance [55, 56, 57] or on associating a single value to each trace [58], the testing framework presented in this paper provides a richer way to express what is *allowed*, *partially allowed*, or *completely forbidden* in non-strict (temporal) requirements.

## 7. CONCLUSIONS AND FUTURE WORK

This paper presented a novel framework to specify and test timed systems showing both soft and hard deadlines. Nine conformance relations, that take into account the different considerations of what a *slightly* erroneous system is, were introduced and the precise relation between these relations was studied. The paper develops a testing theory by introducing a notion of test and by defining how tests are applied to implementations and what is the meaning of passing a test. Finally, it has been proved that testing a system with the appropriate test suite is equivalent to establishing that it is related with the specification from which the test suite was derived.

There is still some room for future work. In particular, this paper is the first step, together with the paper [26], to define a testing theory for systems presenting both time and probabilistic information expressed by means of intervals. A second direction for further research is to increase the expressive power of the formalism. In this line, the formalism [35] can be used to extend the machines used in this paper with data.

*Acknowledgements*

The authors would like to thank the anonymous reviewers of this paper for the careful reading and the useful suggestions to improve the quality of the paper.

## REFERENCES

1. Clarke E, Grumberg O, Peled D. *Model Checking*. MIT Press, 2000.
2. Baier C, Katoen JP. *Principles of Model Checking*. MIT Press, 2008.
3. Myers G. *The Art of Software Testing*. 2nd edn., John Wiley and Sons, 2004.
4. Ammann P, Offutt J. *Introduction to Software Testing*. Cambridge University Press, 2008.
5. Brinksma E, Tretmans J. Testing transition systems: An annotated bibliography. *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, Springer, 2001; 187–195.
6. Hierons R, Bowen J, Harman M ( (eds.)). *Formal Methods and Testing, LNCS 4949*. Springer, 2008.
7. Hierons R, Bogdanov K, Bowen J, Cleaveland R, Derrick J, Dick J, Gheorghe M, Harman M, Kapoor K, Krause P, *et al.*. Using formal methods to support testing. *ACM Computing Surveys* 2009; **41**(2).
8. Rodríguez I. A general testability theory. *20th Int. Conf. on Concurrency Theory, CONCUR'09, LNCS 5710*, Springer, 2009; 572–586.
9. Bergstra J, Ponse A, Smolka S ( (eds.)). *Handbook of Process Algebra*. North Holland, 2001.
10. Brauer W, Reisig W, Rozenberg G ( (eds.)). *Petri Nets I: Central Models and Their Properties, LNCS 254*. Springer, 1987.
11. Brauer W, Reisig W, Rozenberg G ( (eds.)). *Petri Nets II: Applications and Relationships to Other Models of Concurrency, LNCS 255*. Springer, 1987.
12. Petri C, Reisig W. Petri net. *Scholarpedia* 2008; **3**(4):6477.
13. Sifakis J. Use of Petri nets for performance evaluation. *3rd Int. Symposium on Measuring, Modelling and Evaluating Computer Systems*, North-Holland, 1977; 75–93.
14. Zuberek W. Timed Petri nets and preliminary performance evaluation. *7th Annual Symposium on Computer Architecture*, ACM Press, 1980; 88–96.
15. Reed G, Roscoe A. A timed model for communicating sequential processes. *Theoretical Computer Science* 1988; **58**:249–261.
16. Nicollin X, Sifakis J. An overview and synthesis on timed process algebras. *3rd Int. Conf. on Computer Aided Verification, CAV'91, LNCS 575*, Springer, 1991; 376–398.



DOI: 10.1002/stvr

17. Glabbeek Rv, Smolka S, Steffen B. Reactive, generative and stratified models of probabilistic processes. *Information and Computation* 1995; **121**(1):59–80.
18. Baeten J, Middelburg C. *Process algebra with timing*. EATCS Monograph, Springer, 2002.
19. Bravetti M, Aldini A. Discrete time generative-reactive probabilistic processes with different advancing speeds. *Theoretical Computer Science* 2003; **290**(1):355–406.
20. Núñez M. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming* 2003; **56**(1–2):117–177.
21. Springintveld J, Vaandrager F, D'Argenio P. Testing timed automata. *Theoretical Computer Science* 2001; **254**(1-2):225–257. Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.
22. Higashino T, Nakata A, Taniguchi K, Cavalli A. Generating test cases for a timed I/O automaton model. *12th Int. Workshop on Testing of Communicating Systems, IWTCS'99*, Kluwer Academic Publishers, 1999; 197–214.
23. Fecko M, Uyar M, Duale A, Amer P. A technique to generate feasible tests for communications systems with multiple timers. *IEEE/ACM Transactions on Networking* 2003; **11**(5):796–809.
24. En-Nouaary A, Dssouli R. A guided method for testing timed input output automata. *15th Int. Conf. on Testing Communicating Systems, TestCom'03, LNCS 2644*, Springer, 2003; 211–225.
25. Brandán Briones L, Brinksma E. Testing real-time multi input-output systems. *7th Int. Conf. on Formal Engineering Methods, ICFEM'05, LNCS 3785*, Springer, 2005; 264–279.
26. López N, Núñez M, Rodríguez I. Specification, testing and implementation relations for symbolic-probabilistic systems. *Theoretical Computer Science* 2006; **353**(1–3):228–248.
27. Cheung L, Stoelinga M, Vaandrager F. A testing scenario for probabilistic processes. *Journal of the ACM* 2007; **54**(6):Article 29.
28. Merayo M, Núñez M, Rodríguez I. Formal testing from timed finite state machines. *Computer Networks* 2008; **52**(2):432–460.
29. Götz N, Herzog U, Rettelbach M. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. *16th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation, PERFORMANCE'93, LNCS 729*, Springer, 1993; 121–146.
30. Hillston J. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
31. Harrison P, Strulo B. SPADES – a process algebra for discrete event simulation. *Journal of Logic Computation* 2000; **10**(1):3–42.
32. Bravetti M, Gorrieri R. The theory of interactive generalized semi-Markov processes. *Theoretical Computer Science* 2002; **282**(1):5–32.
33. López N, Núñez M, Rubio F. An integrated framework for the analysis of asynchronous communicating stochastic processes. *Formal Aspects of Computing* 2004; **16**(3):238–262.
34. Alur R, Dill D. A theory of timed automata. *Theoretical Computer Science* 1994; **126**:183–235.
35. Merayo M, Núñez M, Rodríguez I. Extending EFSMs to specify and test timed systems with action durations and timeouts. *IEEE Transactions on Computers* 2008; **57**(6):835–848.
36. Merayo M, Núñez M, Hierons R. Testing timed systems modeled by stream X-machines. *Software and Systems Modeling (to appear)* 2010; .
37. Merayo M, Núñez M, Rodríguez I. Formal testing of systems presenting soft and hard deadlines. *2nd IPM Int. Symposium on Fundamentals of Software Engineering, FSEN'07, LNCS 4767*, Springer, 2007; 160–174.
38. Glabbeek Rv. The linear time-branching time spectrum II. The semantics of sequential processes with silent moves. *4th Int. Conf. on Concurrency Theory, CONCUR'93, LNCS 715*, Springer, 1993; 66–81.
39. Tretmans J. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools* 1996; **17**(3):103–120.
40. Núñez M, Rodríguez I. Encoding PAMR into (timed) EFSMs. *22nd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'02, LNCS 2529*, Springer, 2002; 1–16.
41. Francez N. *Fairness*. Springer, 1986.
42. Quemada J, de Frutos D, Azcorra A. TIC: A TImed Calculus. *Formal Aspects of Computing* 1993; **5**:224–252.
43. Nicollin X, Sifakis J. The algebra of timed process, ATP: Theory and application. *Information and Computation* 1994; **114**(1):131–178.
44. Hennessy M, Regan T. A process algebra for timed systems. *Information and Computation* 1995; **117**(2):221–239.
45. Davies J, Schneider S. A brief history of timed CSP. *Theoretical Computer Science* 1995; **138**:243–271.
46. Reed G, Roscoe A. The timed failures – stability model for CSP. *Theoretical Computer Science* 1999; **211**(1–2):85–127.
47. Llana L, Núñez M. Testing semantics for RTPA. *Fundamenta Informaticae* 2009; **90**(3):305–335.
48. de Nicola R, Hennessy M. Testing equivalences for processes. *Theoretical Computer Science* 1984; **34**:83–133.
49. Hennessy M. *Algebraic Theory of Processes*. MIT Press, 1988.
50. Uyar M, Batth S, Wang Y, Fecko M. Algorithms for modeling a class of single timing faults in communication protocols. *IEEE Transactions on Computers* 2008; **57**(2):274–288.
51. Cleaveland R, Lee I, Lewis P, Smolka S. A theory of testing for soft real-time processes. *8th Int. Conf. on Software Engineering and Knowledge Engineering, SEKE'96*, 1996; 474–479.
52. Yuen S, Cleaveland R, Dayar Z, Smolka S. Fully abstract characterizations of testing preorders for probabilistic processes. *5th Int. Conf. on Concurrency Theory, CONCUR'94, LNCS 836*, Springer, 1994; 497–512.
53. Bernardo M, Cleaveland W. A theory of testing for markovian processes. *11th Int. Conf. on Concurrency Theory, CONCUR'2000, LNCS 1877*, Springer, 2000; 305–319.
54. López N, Núñez M. A testing theory for generally distributed stochastic processes. *12th Int. Conf. on Concurrency Theory, CONCUR'01, LNCS 2154*, Springer, 2001; 321–335.
55. Bohnenkamp H, Stoelinga M. Quantitative testing. *8th ACM & IEEE Int. Conf. on Embedded software, EMSOFT'08*, ACM Press, 2008; 227–236.
56. Henzinger T, Majumdar R, Prabhu V. Quantifying similarities between timed systems. *3rd Int. Conf. on Formal Modeling and Analysis of Timed Systems, FORMATS'05, LNCS 3829*, Springer, 2005; 226–241.




DOI: 10.1002/stvr

57. de Alfaro L, Faella M, Stoelinga M. Linear and branching metrics for quantitative transition systems. *31st Int. Colloquium on Automata, Languages and Programming, ICALP'04, LNCS 3142*, Springer, 2004; 97–109.
58. Chatterjee K, Doyen L, Henzinger T. Quantitative languages. *22nd Int. Workshop on Computer Science Logic, CSL'08, LNCS 5213*, Springer, 2008; 385–400.



DOI: 10.1002/stvr