Generation of optimal finite test suites for timed systems[*]

Mercedes G. Merayo, Manuel Núñez and Ismael Rodríguez
Dept. Sistemas Informáticos y Computación
Universidad Complutense de Madrid, E-28040 Madrid. Spain.
e-mail: mgmerayo@fdi.ucm.es, {mn,isrodrig}@sip.ucm.es

Abstract

One of the main problems to test timed systems is that the tester has to decide when to apply the next input to the system under test. Even though the tester could determine good sequences of inputs to find a big variety of errors, the quality of the test suite usually depends on the time when the different parts of the sequences are applied. In this paper we give a formal methodology to provide good time values to test timed systems. These values are computed by taking into account the time stability of the system, that is, if the system is more likely to remain in its current internal state during a given time interval then no input will be applied during that period. In other words, our method will (probabilistically) find those time values that are closer to a change of state in the system, being these values more suitable to apply the appropriate input to the system.

1 Introduction

We can informally describe *timed systems* as those systems where we are not concerned only with *what* the system does but also *when* the system performs its tasks. In order to validate the behavior of this kind of systems, formal methods can play an important role. Thus, formal techniques, in particular, formal specification languages, have being progressively extended to cope with temporal requirements (e.g. [19, 18, 21, 15, 1, 10, 5, 2] among many others). In particular, research on formal testing of timed systems has been very active during the last years (e.g. [20, 11, 16, 8, 7, 12, 3, 14]).

Unfortunately, to *completely* test any non-trivial system is impossible. The problem is that we can invest only a finite amount of time to test a system that can present, in general, infinite different behaviors, being each of them arbitrarily long. Thus, formal testing methodologies usually take certain assumptions to reduce the *testing space*, that is, the different behaviors that must be checked to have enough confidence in the correctness of the system under test (in the following, SUT). Actually, these assumptions may reduce the testing space to be finite. For example, it is very common to assume that the number of states describing the SUT is known and finite. Other assumptions allow to get the testability of the SUT in the *limit*, that is, to use a finite amount of time to test the behavior of the SUT until a given bound (e.g. to consider only the evolutions of the system performing at most n actions). Among these assumptions we may mention to consider that the SUT is deterministic or, alternatively, to consider the *fairness assumption*: After trying *enough* times each of the non-deterministic possibilities, all of them will be performed at least once.

However, in the case of timed systems we cannot usually ensure even *finite testability* in the limit. Even though testing timed systems shares some methods and approaches with testing non-timed ones, it has its own intrinsic difficulties. Let us consider a *bounded behavior* of the SUT and let us discuss what are the new problems. In the case of a timed system, to bound the analysis of the SUT requires not only to limit the amount of inputs that the system may perform but also the time that it takes to apply all these inputs.[1] Let us suppose that we want to test whether the behavior of the SUT is correct for sequences with at most n inputs to be produced in at most t time units. Since the reaction to each input may depend on the point of time when the input is applied to the system, in order to decide about the correction of the SUT we should check its behavior for every possible application of the inputs in the interval $[0, t]$. If we are assuming dense time then we have to try an infinite amount of time values. On the contrary, if we assume a discrete time domain, with a minimum time quantum δ, then the possible number of options will be astronomic as long as we consider values of δ small enough to accurately approximate reality. Thus, the number of tests needed to test the behavior of a timed system in its first t time units is either infinite or very

[*]Research partially supported by the Spanish MEC project WEST TIN2006-15578-C02-01, the Junta de Castilla-La Mancha project PAC06-0008, and the Marie Curie RTN *TAROT* (MRTN-CT-2003-505121).

[1]This example already shows that we have in mind a finite state machine model to describe systems: Each input is followed by an output.

big. In other words, to completely test a timed system is not possible even *in the limit*, that is, to completely test the behavior of a SUT in its first t time units cannot be done in general.

Regardless of the devastating situation described in the previous paragraph, to test timed systems is a must in industrial environments. Thus, researchers continue working on different proposals to overcome the problem. Some approaches propose algorithms to apply test suites covering all the possibilities while testing the SUT. These suites can serve as the basis for future methodologies where not all the possibilities are exercised while testing a timed system but only a *good* subset of them. As discussed before, covering all possibilities implies the generation of either an infinite test suite (e.g. [3, 17]) or a very big one (e.g. [20]). In the latter case, finiteness is usually achieved by considering additional assumptions such that it is possible to discretize continuous time by using a very small, realistic, time quantum while the analysis goes through all the possible (after discretization) time values. Thus, if the time quantum is small enough, the derived test suites are huge. Let us remark that these approaches are somehow inherently contradictory since they start by discretizing the time domain, to reduce the time values to be considered, but they do not make any effort to further reduce the time values to be considered. Another approach to reduce the size of the test suite consists in giving up a rigorous analysis. The idea is to consider that if the finite test suite is *well chosen* then these tests will be able to detect a broad variety of errors in the SUT. This is the approach presented in [4, 13]. Unfortunately, the criteria to choose *good* tests are essentially based on the experience of the tester and in some predefined heuristics.

In this paper we show that the criterium to *choose well* can be more rigorously defined. Given a specification, we would like to derive a test suite such that if the SUT successfully passes these tests in the considered bounded selection (e.g. for all the behaviors having less than n inputs and being performed in less than t time units) then we have the maximal probability of having a correct system. In other words, for any other test suite having the same number of inputs, the probability of finding an error in the SUT is smaller than by using the chosen test suite. Even though to find such *good* test suite is impossible, because we cannot infer anything about all the infinite behaviors that are not analyzed, we can get such a test suite if we add a *hypothesis* about the SUT that, we think, is reasonable.

Next, we informally describe our approach. Let us consider the execution of a timed system during t time units. Our systems can perform either *usual* actions, that is, inputs and outputs, or time transitions, associated with timeouts. In Figure 1 we show a graphical representation of our systems. s_1 is the initial step; a transition $s \xrightarrow{i/o} s'$ means that if the machine is placed in the state s and receives the input i then it will produce the output o and change its state to s'; a transition $s \xrightarrow{t} s'$ means that if the machine is in state s and does not receive an input before t time units then the machine will change its state to s'. If we are testing the behavior of such a system, it would be very useful to know the internal state of the SUT. However, we consider a black-box testing approach. So, we can only try to infer the state of the machine by analyzing its responses to tests (i.e. sequences of inputs). The problem is that these systems can also evolve by performing timeouts which are not *visible*. Thus, we have to *hypothesize* about the possibility of the SUT performing a timeout. While testing a SUT we have to take into account what the specification expects. For example, let us consider the specification depicted in Figure 1 and a given SUT such that we are testing its correctness with respect to this specification. If the SUT is placed at its initial state, we would expect that if we apply the input i' at any time before 3 time units pass then we should receive the output o'. The problem is that we cannot apply i' in all the possible values belonging to the interval $[0, 3]$. In this paper we provide a methodology to choose the *best* time values in this interval so that we can (partially) overcome the previously exposed limitations while testing timed systems.

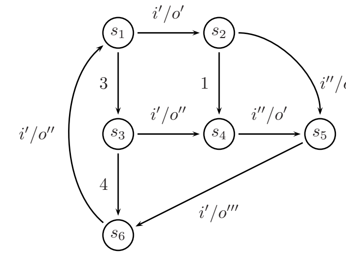

Figure 1. Representation of `TFSM`**s.**

As we indicated before, we cannot observe when a timeout is performed. In probabilistic terms, the probability that a timeout has been performed in the last t' time units obviously increases with t'. Let us assume that we have a function returning these probabilities, that is, if a system is in state s then $f(s, t')$ returns the probability that the system did not perform a timeout in the last t' time units. Let us consider that at time t we offer the input a and the system replies with output b.Thus, the probability that t' time units before the system would have produced the output b after receiving the input a is at least $f(s, t')$. This is so because if the system did not change its state then it would produce b after receiving a; if the system would have changed its state (this happens with probability $1 - f(s, t')$) then it could still produce b (e.g., because it moved to another state with the same behavior). In general, by using such a probability function, we can compute a probabilistic lower bound with which the system would have produced, in any time before the current time, the output b after receiving the input a. This probabilistic information is specially relevant since by applying an input at a specific point of time we can infer not only the behavior of the SUT at that precise point of time but also, in probabilistic terms, the output that would have been received at previous points of time. So, we can choose which time values are *better* to apply inputs in the sense that, with a big probability, *nothing happened* before inside the system, that is, the SUT did not evolve by performing a time transition associated with a timeout.

The consideration of the previous property in testing terms allows to obtain a complete (probabilistic) coverage in a bounded selection, even if infinite time values are considered, by applying a finite test suite. Let $\mathtt{size}(\mathcal{T})$ be the number of input applications of the tests belonging to the suite $\mathcal{T}$ and $\mathtt{prob\_ok}(S, \mathcal{T}, t, n)$ be the probability that the SUT S is correct during its first t time units and for behaviors having less than n inputs, after successfully passing all the tests belonging to $\mathcal{T}$. Given $m \in \mathbb{N}$, we can build a test suite $\mathcal{T}_1$, with $\mathtt{size}(\mathcal{T}_1) = m$, such that for any other test suite $\mathcal{T}_2$, with $\mathtt{size}(\mathcal{T}_2) \leq m$, we have $\mathtt{prob\_ok}(S, \mathcal{T}_1, t, n) \geq \mathtt{prob\_ok}(S, \mathcal{T}_2, t, n)$. In this paper we present a formal procedure to choose good tests by following a reasonable and intuitive approach: Choose those tests having the best *representative* time values. These values will be computed by taking into account the function f.

Regarding the definition of f, in the previous informal presentation we skipped some of its properties. For example, it seems reasonable that the *stability* of the SUT depends on its current state. Unfortunately, since we assume that the SUT is a black box, we cannot know the current state of the SUT. Thus, f must depend on information that we are able to extract from the SUT. This information is the sequences that the SUT performs when stimulated by an input sequence. So, we will not have expressions such as $f(s, t)$. During the rest of the paper $\mathcal{F}(t, \sigma)$ will indicate the probability that after performing the sequence $\sigma = (t_1/i_1/o_1, t_2/i_2/o_2, \ldots, t_r/i_r/o_r)$, meaning that for each $1 \leq j \leq r$, we wait t_j time units to apply the input i_j and observe the output o_j, the state is the same as it was t time units before, that is, at time $\sum t_j - t$, being $t \leq t_r$.

Let us remark that to consider $\mathcal{F}$ as a testing hypothesis provided by the tester is only one of the possibilities. Another possibility consists in *computing* it. In order to approximate $\mathcal{F}$ we can take several approaches. As a first approximation, we can consider that the SUT should have, more or less, the same tendency to change the state as the specification. In fact, if the SUT is very different from the specification then we should be able to find an error without using a lot of temporal information. Thus, we can use the tendency of the specification to change the state by performing a time transition to estimate the corresponding probability for the SUT. This technique assumes *presumption of innocence* of the SUT. Another possibility consists in using the experimentation on the SUT to estimate $\mathcal{F}$: According to the SUT responses, we can try to identify those time values such that after applying the same sequence of inputs we receive different outputs. This information is useful to evaluate the variability degree of the SUT, what can be used to estimate $\mathcal{F}$. In order to implement these ideas, we are exploring to derive $\mathcal{F}$ by using *genetic algorithms*. The idea is to reduce the problem of deciding which time values are more likely to produce a timeout to a searching problem. In this line, genetic algorithms and formal testing have been recently combined to produce UIO sequences [9, 6]. These results are very promising but are not yet finished and fall outside the scope of this paper. Anyway, the methodology presented in this paper is independent of whether we have that $\mathcal{F}$ is given or we have to estimate it by using the specification, the responses of the SUT, or a combination of both. Our methodology represents a rigorous and formal approach to choose tests and we think that this is the main contribution with respect to previous work on testing timed systems.

The rest of the paper is structured as follows. In the next section we present our temporal model and we show how tests are applied to systems. In Section 3 we describe a formal method to find test suites with high testing coverage. Finally, in Section 4 we present our conclusions.

2 The Formal Model

In this section we introduce our formalism to specify timed systems and the appropriate notion of test. Next, we will formally define how timed systems are tested. All these concepts will be used in the next section to decide which tests, more specifically, *when* to test, are the ones giving more information about the SUT.

First, we introduce our notion of *timed finite state machine*. As we have indicated in the introduction of the paper, we will add new features so that the timed behavior of a system can be properly specified. We will consider that the machine can evolve not only by performing *usual* actions but also by raising *timeouts*. Intuitively, if after a given time, depending on the current state, we do not receive any input action then the machine will change its current state. In order to simplify the model, we do not consider that the performance of actions take time. However, this can be easily simulated by using timeouts in the appropriate form. In fact, some of the definitions in this section are *adaptations* of the ones introduced in [14] where a model with both timeouts and action durations is presented.

During the rest of the paper we will use the following notation. A tuple of elements $(e_1, e_2 \ldots, e_n)$ will be denoted by $\bar{e}$. $\hat{a}$ denotes an interval of elements $[a_1, a_2)$. A tuple of intervals is denoted by $\check{t}$. Let $\check{q} = (\hat{q_1}, \ldots, \hat{q_n})$ and

$\bar{t} = (t_1, \ldots, t_n)$. We write $\bar{t} \in \check{q}$ if for all $1 \leq j \leq n$ we have $t_j \in \hat{q}_j$. We will use the projection functions π_i such that given a tuple $\bar{t} = (t_1, \ldots, t_n)$, for all $1 \leq i \leq n$ we have $\pi_i(\bar{t}) = t_i$. Let $\bar{t} = (t_1, \ldots, t_n)$ and $\bar{t}' = (t'_1, \ldots, t'_n)$. We write $\bar{t} = \bar{t}'$ if for all $1 \leq j \leq n$ we have $t_j = t'_j$. We write $\bar{t} \leq \bar{t}'$ if for all $1 \leq j \leq n$ we have $t_j \leq t'_j$. Finally, we will denote by $\sum \bar{t}$ the sum of all elements of the tuple $\bar{t}$, that is, $\sum_{j=1}^{n} t_j$.

Definition 1 A *Timed Finite State Machine*, in the following `TFSM`, is a tuple $M = (S, I, O, TO, Tr, s_{in})$ where S is a finite set of *states*, I is the set of *input* actions, O is the set of *output* actions, $TO : S \longrightarrow S \times (\mathbb{R}_+ \cup \infty)$ is the *timeout function*, Tr is the set of *action transitions*, and s_{in} is the *initial state*.

An *action transition* is a tuple (s, s', i, o) where $s, s' \in S$ are the initial and final state of the transition, and $i \in I$, $o \in O$ are the input and output actions associated with the transition.

We say that M is *deterministic* if for all state $s \in S$ and input $i \in I$, $(s, s_1, i, o_1), (s, s_2, i, o_2) \in Tr$ implies $s_1 = s_2$ and $o_1 = o_2$.

We say that M is *input-enabled* if for all state $s \in S$ and input $i \in I$ there exist s' and o such that $(s, s', i, o) \in Tr$. □

An action transition (s, s', i, o) denotes that if the input i is received then the output o will be produced and the machine will change its internal state to s'. We will sometimes denote the transition (s, s', i, o) by $s \xrightarrow{i/o} s'$. For each state $s \in S$, the application of the timeout function $TO(s)$ returns a pair (s', t) indicating the time t that the machine can remain at the state s waiting for an input action, and the state s' to which the machine evolves if no input is received on time. We assume that $TO(s) = (s', t)$ implies $s \neq s'$, that is, timeouts always produce a change of the state. We indicate the absence of a timeout in a given state by setting the corresponding time value to ∞. By abusing the notation, we will assume $r + \infty = \infty$ for all $r \in \mathbb{R}$.

Example 1 In Figure 1 we give a graphical representation of the `TFSM`:

$$M = (\{s_1, \ldots, s_6\}, \{i', i''\}, \{o', o'', o'''\}, TO, Tr, s_1)$$

where Tr and TO are defined as:

$$Tr = \left\{ \begin{array}{l} t_{12} = (s_1, s_2, i', o'),\ t_{34} = (s_3, s_4, i', o''), \\ t_{25} = (s_2, s_5, i'', o'''),\ t_{45} = (s_4, s_5, i'', o'), \\ t_{56} = (s_5, s_6, i', o'''),\ t_{61} = (s_6, s_1, i', o'') \end{array} \right\}$$

$TO(s_1) = (s_3, 3)$, $TO(s_2) = (s_4, 1)$, $TO(s_3) = (s_6, 4)$, and $TO(s) = (s_1, \infty)$, for $s \in \{s_4, s_5, s_6\}$. □

Next we introduce some notions to consider the concatenation of several transitions of a `TFSM`. A *step* is an action transition preceded by zero or more timeouts. The interval $\hat{t}$ indicates the time values where the input action could be received. An *evolution* is a sequence of inputs/outputs corresponding to the action transitions of a chain of steps where the first one begins with the initial configuration of the machine. In addition, evolutions include time values which inform us about possible timeouts (indicated by the intervals $\hat{t}_j$). In some situations, mainly in the forthcoming examples, we will need to know the state in which a machine is placed after letting pass a certain amount of time. Thus, $s \overset{t}{\leadsto} s'$ denotes that if the machine is in state s and t time units pass without performing any action then the machine will be placed in state s'. Let us remark that s and s' will coincide, for instance, if $\pi_2(TO(s)) \geq t$.

Definition 2 Let $M = (S, I, O, TO, Tr, s_{in})$ be a `TFSM` and $s_0 \in S$. A tuple $(s_0, s, i/o, \hat{t})$ is a *step* of M for s_0 if there exist $k \geq 0$ states $s_1, \ldots, s_k \in S$ such that for all $1 \leq j \leq k$ we have $TO(s_{j-1}) = (s_j, t_j)$ and there exists a transition $(s_k, s, i, o) \in T_r$ such that $\hat{t} = \left[\sum_{j=1}^{k} t_j, \sum_{j=1}^{k} t_j + \pi_2(TO(s_k))\right)$. We denote by `Steps`$(M, s)$ the set of steps of M for the state s.

We say that $(\hat{t}_1/i_1/o_1, \ldots, \hat{t}_r/i_r/o_r)$ is an *evolution* of M if there exist r steps of M $(s_{in}, s_1, i_1/o_1, \hat{t}_1), (s_1, s_2, i_2/o_2, \hat{t}_2), \ldots, (s_{r-1}, s_r, i_r/o_r, \hat{t}_r)$ for the states $s_{in}, s_1, \ldots, s_{r-1}$, respectively. We denote by `TEvol`(M) the set of evolutions of M.

Let $s, s' \in S$ and $t \in \mathbb{R}_+$. We write $s \overset{t}{\leadsto} s'$ if there exist $k \geq 0$ states $s_1, \ldots, s_k \in S$ such that for all $1 \leq j \leq k$ we have $TO(s_{j-1}) = (s_j, t_j)$ and $t \in \left[\sum_{j=1}^{k} t_j, \sum_{j=1}^{k} t_j + \pi_2(TO(s_k))\right)$. □

Example 2 Let us consider the `TFSM` depicted in Figure 1 and presented in Example 1. Next, we give some of the *steps* that the machine can generate. For example, $(s_1, s_2, i'/o', [0, 3))$, represents the transition t_{12} when no timeouts precede it. The input i' can be accepted before 3 time units pass (this is indicated by the interval $[0, 3)$). The step $(s_1, s_4, i'/o'', [3, 7))$ is built from the timeout transition associated to the state s_1 and the action transition t_{34}. This step represents that if after 3 time units no input is received, the timeout transition associated with that state will be triggered and the state will change to s_3. After this, the machine can accept the input i' before 4 units of time pass, that is, the timeout assigned to the state s_3. So during the time interval $[3, 7)$ if the machine receives the input i' it will emit o'' and the state will change to s_4. Similarly, we can obtain the step $(s_1, s_1, i'/o'', [7, \infty))$, using the timeout transitions corresponding to s_1 and s_3 and the action transition t_{61}.

For example, $([7, \infty)/i'/o'', [3, 7)/i'/o'')$ represents an evolution built from two steps and assuming that s_1 is the initial state.

Finally, the following are possible time transitions: $s_1 \xrightarrow{2} s_1$, $s_3 \xrightarrow{6} s_6$, $s_1 \xrightarrow{8} s_6$. □

In the following definition we introduce the concept of *instanced evolution*. Intuitively, instanced evolutions are constructed from evolutions by instantiating to a concrete value each timeout, given by an interval, of the evolution. In addition, we need to extract those evolutions that are performed inside a *bound*, that is, they have less than n inputs and take less than t time units.

Definition 3 Let $M = (S, I, O, TO, Tr, s_{in})$ be a $\texttt{TFSM}$ and $e = (\hat{t}_1/i_1/o_1, \ldots, \hat{t}_r/i_r/o_r)$ be an evolution. We say that the tuple $(t_1/i_1/o_1, \ldots, t_r/i_r/o_r)$ is an *instanced evolution of* e if for all $1 \leq j \leq r$ we have $t_j \in \hat{t}_j$. We denote by $\texttt{InsTEvol}(M)$ the set of instanced evolutions of M. By abusing the notation, we will sometimes refer to instanced evolutions such as $(t_1/i_1/o_1, \ldots, t_r/i_r/o_r)$ as $(\bar{t}, \sigma)$, where $\bar{t} = (t_1, \ldots, t_r)$ and $\sigma = (i_1/o_1, \ldots, i_r/o_r)$.

Let $ie = (t_1/i_1/o_1, \ldots, t_r/i_r/o_r)$ be an instanced evolution. We define its *duration* as $\texttt{dur}(ie) = \sum t_i$ and its *length* as $\texttt{len}(ie) = r$. We define the set of *instanced evolutions bounded* by n and t of M, denoted by $\texttt{bound}_M(n, t)$, as

$$\texttt{bound}_M(n,t) = \left\{ ie \in \texttt{InsTEvol}(M) \;\middle|\; \begin{array}{l} \texttt{dur}(ie) \leq t, \\ \texttt{len}(ie) \leq n \end{array} \right\}$$

□

Example 3 If we consider the temporal evolution previously showed, $([7,\infty)/i'/o'', [3,7)/i'/o'')$, we have that $(8/i'/o'', 5/i'/o'')$ and $(12/i'/o'', 3/i'/o'')$ are instanced evolutions. □

Next we define how tests are applied to SUTs. The idea will be that we derive a test suite from the specification and apply its tests to the SUT. If the SUT fails any of the tests then we know that the SUT is not a correct implementation of the specification. We assume, as usual, that SUTs can be formally represented by using the same formalisms as for specification, that is, by means of $\texttt{TFSM}$s. We also take the usual assumption that all the input actions are always enabled in any state of the SUT. The idea is that we need the SUT to reply somehow to any provided stimulus. Finally, in this paper we assume that both specifications and SUTs are deterministic. This restriction is only needed to simplify the presentation of the forthcoming technical machinery. The extension to cope with non-determinism can be done by following [14].

Tests represent sequences of inputs applied to an SUT. Once an output is received, the tester checks whether it belongs to the set of expected ones or not. In the latter case, a fail signal is produced. In the former case, either a pass signal is emitted (indicating successful termination) or the testing process continues by applying another input. If we are testing an implementation with input and output sets I and O, respectively, tests are deterministic acyclic labelled transition systems (i.e. trees) with a strict alternation between an input action and the set of output actions. After an output action we may find either a leaf or another input action. Leaves can be labelled either by *pass* or by *fail*. In addition to check the functional behavior of the SUT, tests have also to detect whether wrong timed behaviors appear. Specifically, tests will include *delays* before offering input actions. The idea is that delays in tests will induce timeouts in SUTs. Thus, we may indirectly check whether the timeouts imposed by the specification are reflected in the SUT by offering input actions after a specific delay. Let us note that a tester cannot observe when the SUT takes a timeout. However, she can check the SUT behavior after different delays.

Definition 4 A *test* is a tuple $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, D)$ where S is the set of states, I and O are disjoint sets of input and output actions, respectively, $Tr \subseteq S \times (I \cup O) \times S$ is the transition relation, $s_0 \in S$ is the initial state, and the sets $S_I, S_O, S_F, S_P \subseteq S$ are a partition of S. The transition relation and the sets of states fulfill the following conditions:

- S_I is the set of *input* states. We have that $s_0 \in S_I$. For all input state $s \in S_I$ there exists a unique outgoing transition $(s, a, s') \in Tr$. For this transition we have that $a \in I$ and $s' \in S_O$.
- S_O is the set of *output* states. For all output state $s \in S_O$ we have that for all $o \in O$ there exists a unique state s' such that $(s, o, s') \in Tr$. In this case, $s' \notin S_O$. Moreover, there do not exist $i \in I$ and $s' \in S$ such that $(s, i, s') \in Tr$.
- S_F and S_P are the sets of *fail* and *pass* states, respectively. We say that these states are *terminal*. Thus, for all state $s \in S_F \cup S_P$ we have that there do not exist $a \in I \cup O$ and $s' \in S$ such that $(s, a, s') \in Tr$.

Finally, we have the function $D : S_I \longrightarrow \mathbb{R}_+$ associating delays with input states.

We say that a test T is *valid* if the graph induced by T is a tree with root at the initial state s_0.

Let $\sigma = i_1/o_1, \ldots, i_r/o_r$. We write $T \xrightarrow{\sigma} s$ if $s \in S_F \cup S_P$ and σ leads T to the state s. More formally, 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 $2 \leq j \leq r$ we have $(s_{j1}, i_j, s_{j2}) \in Tr$, and for all $1 \leq j \leq r-1$ we have $(s_{j2}, o_j, s_{(j+1)1}) \in Tr$. Let $\bar{t} = (t_1, \ldots, t_r)$. We write $T \xrightarrow{\sigma}_{\bar{t}} s$ if $T \xrightarrow{\sigma} s$, $t_1 = D(s_0)$, and for all $1 < j \leq r$ we have $t_j = D(s_{j1})$. □

Let us remark that $T \xrightarrow{\sigma} s^T$, and its variant $T \xrightarrow{\sigma}_{\bar{t}} s^T$, imply that s is a terminal state. Next we define the application of a test suite to a SUT. We say that the test suite $\mathcal{T}$ is *passed* if for all test the terminal states reached by the composition of the SUT and the test are *pass* states.

Definition 5 Let I be a TEFSM, T be a valid test, $\sigma = i_1/o_1, \dots, i_r/o_r$, s^T be a state of T, and $\bar{t} = (t_1, \dots, t_r)$. We write $I \parallel T \xrightarrow{\sigma}_{\bar{t}} s^T$ if $T \xrightarrow{\sigma}_{\bar{t}} s^T$ and $(\bar{t}, \sigma) \in \mathtt{InsTEvol}(I)$.

We say that I *passes* the set of valid tests $\mathcal{T}$, denoted by $\mathtt{pass}(I, \mathcal{T})$, if for all test $T \in \mathcal{T}$ there do not exist σ, s^T, $\bar{t}$ such that $I \parallel T \xrightarrow{\sigma}_{\bar{t}} s^T$ and $s^T \in S_F$. $\square$

We conclude this section by providing a formal definition of the probabilistic functions $\mathcal{F}$ that we presented in the introduction of the paper.

Definition 6 Let M be a TFSM.We say that a function $\mathcal{F}_M : \mathbb{R}_+ \times \mathtt{InsTEvol}(M) \longrightarrow [0, 1]$ is a *probabilistic stability function* for M. $\square$

As we indicated in the introduction, given a time t and an instanced evolution $ie = (t_1/i_1/o_1, \dots, t_r/i_r/o_r)$, $\mathcal{F}_M(t, ie)$ denotes the probability that, after performing ie, the machine M has not changed its state in the last t time units (i.e., it has not performed a timeout in the last t time units). Besides, we will assume that this function can be used in the opposite direction as well: It also denotes the probability that if i_r would have been offered t time units *later* then the state would have not changed from the time i_r was actually offered to this time.[2]

3 Computing the coverage of a test suite

In this section we deploy the machinery to establish the goodness of a test suite. First, we informally introduce our framework by using a simple example. Let us suppose that the we apply the test "wait two seconds and then offer the input a" and we observe the output b. Then, we wait two more seconds, apply a, and observe again b. We repeat the operation again and observe the same result. In other words, we observe the sequence

$$s_1 \overset{2}{\rightsquigarrow} s_2 \xrightarrow{a/b} s_3 \overset{2}{\rightsquigarrow} s_4 \xrightarrow{a/b} s_5 \overset{2}{\rightsquigarrow} s_6 \xrightarrow{a/b} s_7$$

for some unknown, possibly equal, states $s_1, \dots, s_7$. In addition, let us suppose that the function $\mathcal{F}$, associated with the testing process, indicates that, for all traces, the state of the SUT did not change in the last second (i.e., a timeout was not performed in the last second) with probability $\frac{n}{n+1}$, being n equal to the length (i.e. number of inputs) of the trace. According to all the information that we have, let us compute the probability to obtain three bs after applying three as but having *one* second delays (instead of two).

[2] We could use two different functions, one for each purpose. However, we preferred to use a single function for the sake of presentation simplicity.

Let us consider the first application of a in the before mentioned test, that is, two time units passed, we applied a and then we observed b. If we would have waited only one second instead of two, then we would have $s_1 \overset{1}{\rightsquigarrow} s_2'$ for some state s_2'. At this point, the probability of being at the same state as the one reached after waiting for two seconds (that is, of $s_2 = s_2'$) would be at least $\frac{1}{2}$: Since we use the sequence $(2/a/b)$ to infer what would happen if a were offered one second before, we have $\frac{1}{1+1}$ according to $\mathcal{F}$. Thus, the probability of observing b in the SUT after offering a is at least $\frac{1}{2}$. In the previous argument we use the clause "at least" because this probability can be bigger. For example, a timeout could have been performed bringing the machine to a state that produces b after applying a. Another possible situation is that several timeouts were performed during the last second so that we reach the same state where we were one second ago. Obviously, in this situation we would also receive b after applying a.

What would happen *next* if we apply a again after waiting only one more second? In this case, until the new a is offered we we would have the sequence

$$s_1 \overset{1}{\rightsquigarrow} s_2' \xrightarrow{a/b} s_3' \overset{1}{\rightsquigarrow} s_4'$$

for some unknown, possibly equal, states s_1, s_2', s_3', s_4'. If the state s_3' in this sequence and the state s_3 reached by applying our test were actually the same then we could apply an argument similar to the previous one to conclude that with a probability at least $\frac{2}{2+1} = \frac{2}{3}$ we will obtain b after a. This is so because $\mathcal{F}$ is applied to a trace of length 2. So, what is the probability of $s_3 = s_3'$? This probability is again at least $\frac{1}{2}$. This is so because, as previously explained, this is the lower bound of the probability of $s_2 = s_2'$. In conclusion, the probability that after applying two as, each preceded by a one second delay, we obtain two bs is at least $\frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$. Using a similar reasoning, the probability that we obtain three bs after applying three as, each of them delayed one second is greater than $\frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} = \frac{1}{4}$.

Let us note that observations and $\mathcal{F}$ can be used to extrapolate the behavior in time values previous to the observation (like in the previous example) or in *subsequent* time values. In the former case, after we apply i and observe o at a given time, we hypothesize about what would happen if i would have been offered before. In the latter, we hypothesize about what would happen if i would have been offered later. For example, we can use arguments similar to those of the previous example to infer the probability that 3 bs would be produced if 3 as were offered at delays 3, 3, and 3, or at delays 1, 3, and 1, etc.

The running example commented at the beginning of the section considers that we have only one test application. However, we have to consider the application of a test suite. How can we use the multiple executions if we obtain different observation time values? For example, let us suppose that we have two different test applications and we want to infer what would happen if we offer a after a one second delay. Let us consider that the test applications show b when offering a after a two (resp. three) seconds delay. In this case, we will take as probabilistic bound the higher value. If $\mathcal{F}$ is a decreasing monotonous function, as expected, then we use our observation at time two to infer what would happen at time one, and we ignore the probability at time three.

The previously introduced method will be the basis to compute a notion of test suite coverage, that is, how *good* was the SUT tested (or how good is a given *test suite* to provide such a coverage). However, in order to compute this measure, it will not be enough to use the probabilistic knowledge about one specific trace but we have to take into account *all* the possible executions that fit into our testing constraints (i.e., with less than n inputs and taking less than t units of time). By adding the probabilities associated with each of these executions we obtain a probabilistic bound on the correctness of the SUT. Let us remark that if the time domain is dense, to *add* probabilities becomes to *integrate*. In order to obtain these probabilities, we will apply the $\mathcal{F}$ function to the corresponding tests as explained before. Actually, for each evolution in the specification we will add (i.e., integrate) the probabilities of all instanced evolutions fitting into the evolution. We will repeat this process for all the evolutions of the specification having less than n inputs and lasting less than t time units. Let us remark that the number of such evolutions is finite. Then, by considering together all these probabilities (the ones computed for each evolution from its instanced evolutions), we will have a probabilistic measure on the correctness of the SUT: A lower bound on the aggregated probability that the SUT provides, after each sequence of inputs, the outputs expected by the specification.

Next we show how the probability of observing a given sequence of outputs in response to a sequence of inputs at some given times is extracted from $\mathcal{F}$ and a finite set of observations collected from the SUT. Basically, we will search for observations of the set where the same sequences of inputs/outputs are produced but possibly at different times. One of these observations will be selected, and the target probability will be extracted from $\mathcal{F}$ by taking into account the difference between each time of the sequence and the corresponding time in the observation. The observation selected for making this calculation will be the one providing the highest probability, according to the conditions commented before.

Let us note that even though the term *observation* refers to a sequence observed by testing, we can reason about these observations *before* tests are applied. Due to the determinism of the specification, there is a single *valid* response for each sequence of inputs produced at specific times (let us note that tests have a single *pass* state in this case). Hence, if the SUT passes a test then the set of valid observations that can be collected is a singleton. Let us remark that our goal is to select those tests such that, if all of them are *passed*, then they provide the highest probability of correctness of the SUT. Hence, in order to reason about test suites it is enough to reason about the sets of observations that would be collected from them. Hence, our method can be easily applied to measure the coverage of tests even *before* they are applied. In the next definition, the set of observations (denoted by $\mathcal{O}$) is represented by a set of instanced evolutions. Basically, we define a function associating each instanced evolution with the probability that its outputs are produced in response to its inputs, according to $\mathcal{F}$ and the set of observations $\mathcal{O}$.

Definition 7 Let $M = (S, I, O, TO, Tr, s_{in})$ be a $\texttt{TFSM}$, $\mathcal{O} \subseteq \texttt{InsTEvol}(M)$ be a set of instanced evolutions, and $\mathcal{F}_M$ be a probabilistic stability function for M. The *extrapolated probabilistic behavior function* associated to O and $\mathcal{F}$, denoted by $P_{\mathcal{O},\mathcal{F}}$, is a function $P_{\mathcal{O},\mathcal{F}} : \texttt{InsTEvol}(M) \longrightarrow [0,1]$ such that for all $ie = (t_1/i_1/o_1, \ldots, t_n/i_n/o_n) \in \texttt{InsTEvol}(M)$ we have

$$P_{\mathcal{O},\mathcal{F}}(ie) = \texttt{max}\{P'(ob) |\ ob \in \mathcal{O}\}$$

where for all $ob = (t'_1/i_1/o_1, \ldots, t'_n/i_n/o_n) \in \texttt{InsTEvol}(M)$ we have

$$P'(ob) = \prod_{i=1}^{n} \mathcal{F}(|t_i - t'_i|, (t'_1/i_1/o_1, \ldots, t'_i/i_i/o_i))$$

□

Next we show an example to illustrate this definition. Given $r \in \mathbb{R}$, we will assume that $\diamondsuit(r) = r$ if $0 \leq r \leq 1$, $\diamondsuit(r) = 0$ if $r < 0$, and $\diamondsuit(r) = 1$ if $r > 1$. This function will be used to transform any $r \in \mathbb{R}$ into a valid probability in the interval $[0, 1]$.

Example 4 Let us revisit the informal example given at the beginning of this section. Using the notation introduced in the previous definition we have $\mathcal{O} = \{(2/a/b, 2/a/b, 2/a/b)\}$. A function $\mathcal{F}$ fitting into the description given at the beginning of the section is $\mathcal{F}(t, \sigma) = \diamondsuit\left(\frac{n}{n+1} - t + 1\right)$, where n is the length of σ (for the sake of notation simplicity, the application of the function $\diamondsuit$ will be omitted from now on). Let us note that the value $\frac{n}{n+1}$ is returned when $t = 1$. According to this definition of $\mathcal{F}$, for all $\sigma = (t_1/a/b, t_2/a/b, t_3/a/b)$ we have that $P_{\mathcal{O},\mathcal{F}}(\sigma)$ is equal to:

$$\left(\frac{1}{2} - |2 - t_1| + 1\right) \cdot \left(\frac{2}{3} - |2 - t_2| + 1\right) \cdot \left(\frac{3}{4} - |2 - t_3| + 1\right)$$

For instance, $P_{\mathcal{O},\mathcal{F}}(1/a/b, 1/a/b, 1/a/b) = \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4}$.

Let us note that, since there are not observations in $\mathcal{O}$ concerning sequences of inputs/outputs different from $(a/b, a/b, a/b)$, we have that $P_{\mathcal{O},\mathcal{F}}$ returns 0 for any instanced evolution with other inputs/outputs. The reason is that the `max` function used in the previous definition returns 0 if applied to an empty set. □

We are not interested in the probability of an instanced evolution but in the probability of *all* of them. In order to be able to range them, we need to infer all the time values where, according to the specification, a given sequence could be produced. For each evolution we will collect all the available time values for each step, that is, such that an instanced evolution of the evolution is obtained. Moreover, since we are interested in testing the SUT up to a given time limit t, only instanced evolutions lasting less than t will be considered. Given an evolution and $i-1$ real numbers denoting the *concrete* time values at which the first $i-1$ inputs are offered, the next function computes the values at which the i-th input could be produced, according to the limit imposed by t.

Definition 8 Let $M = (S, I, O, TO, Tr, s_{in})$ be a `TFSM`, $e = (\hat{t}_1/i_1/o_1, \ldots, \hat{t}_n/i_n/o_n) \in \texttt{TEvol}(M)$ be an evolution of M, $i \in [1..n]$, and $t \in \mathbb{R}_+$. The *range function* for the i-th step of e with time limit t, denoted by $R_{i,e,t}$, is a function $R_{i,e,t} : (\mathbb{R}_+)^{i-1} \longrightarrow \mathbb{R}_+ \times \mathbb{R}_+$ where for all $t_1, \ldots, t_{i-1} \in \mathbb{R}_+$ we have

$$R_{i,e,t}(t_1, \ldots, t_{i-1}) = \Big[\, t_{i1} \ , \ \texttt{min}\{t_{i2}, t - \textstyle\sum_{j=1}^{i-1} t_j\} \Big)$$

where $\hat{t}_i = [t_{i1}, t_{i2})$. If $i = 1$ then the value returned by $R_{i,e,t}$ will be denoted by $R_{i,e,t}(\cdot)$. □

Next we show how to add the probability of correctness of each instanced evolution fitting into an evolution, provided that it is executed before t units of time.

Definition 9 Let M be a `TFSM`, $\mathcal{F}$ be a probabilistic stability function for M, $t \in \mathbb{R}_+$, and $e = (\hat{t}_1/i_1/o_1, \ldots, \hat{t}_n/i_n/o_n) \in \texttt{TEvol}(M)$ be an evolution of M. Besides, let $\mathcal{O} \subseteq \texttt{InsTEvol}(M)$ be a set of instanced evolutions of M. We define the *cumulated probability* associated with e as

$$CP_{\mathcal{O},\mathcal{F}}(e) = \int^{R_{1,e,t}(\cdot), \ldots, R_{n,e,t}(t_1, \ldots, t_{n-1})} P_{\mathcal{O},\mathcal{F}}(t_1/i_1/o_1, \ldots, t_n/i_n/o_n)\ d\mathbf{t_1} \ldots d\mathbf{t_n}$$

□

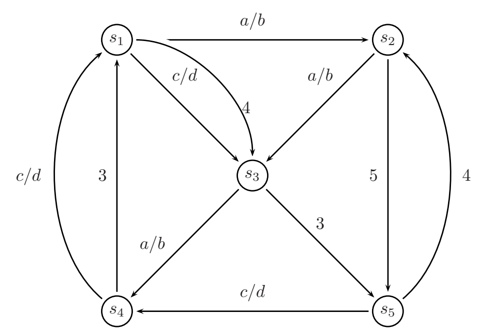

Figure 2. Specification given as a `TFSM`.

Intuitively, $CP_{\mathcal{O},\mathcal{F}}$ computes the probability associated with all the possible instanced evolutions of the evolution e. The limits of the n integrals are given by the intervals provided by the $R_{i,e,t}$ function. That is, the integration limits for the variable t_1 are given by the interval denoted by $R_{1,e,t}$, the limits for t_2 are given by $R_{2,e,t}$, and so on.

Example 5 Let us suppose that we are testing a SUT with respect to the specification M given in Figure 2 up to time $t = 10$, assuming that s_1 is its initial state. As in Example 4, let us suppose that we are provided with the set of observations $\mathcal{O} = \{(2/a/b, 2/a/b, 2/a/b)\}$ and the same function $\mathcal{F}$ is used. Let us consider the evolution $e = ([0,4)/a/b, [0,5)/a/b, [0,3)/a/b) \in \texttt{TEvol}(M)$. We will add the probability of correctness of all instanced evolutions associated to e.

First, let us compute the time intervals we will have to take into account to cover all instanced evolutions fitting into e such that they are produced in at most 10 seconds. By applying Definition 8, we infer that the interval of the first delay is given by $R_{1,e,10}(\cdot) = [0,4)$, the second interval is $R_{2,e,10}(t_1) = [0,5)$, and the third interval is defined by the expression $R_{3,e,10}(t_1,t_2) = [0, \texttt{min}\{3, (10 - (t_1 + t_2))\})$. These intervals will define the integration limits used in the computation of CP.

In Example 4 we computed the function $P_{\mathcal{O},\mathcal{F}}$ for all instanced evolutions of the form $(t_1/a/b, t_2/a/b, t_3/a/b)$. According to it, the cumulated probability of all instanced evolutions associated to e that can be produced in at most 10 time units, denoted by $CP_{\mathcal{O},\mathcal{F}}(e)$, is given by

$$\int_0^{t'}\int_0^5\int_0^4 \left(\frac{1}{2} - |2 - t_1| + 1\right) \cdot \left(\frac{2}{3} - |2 - t_2| + 1\right) \cdot \left(\frac{3}{4} - |2 - t_3| + 1\right)\ dt_1\, dt_2\, dt_3$$

where $t' = \texttt{min}\{3, (10 - (t_1 + t_2))\}$.

Alternatively, let us consider now that the set of observations $\mathcal{O}$ is given by the expression

$\{(2/a/b, 2/a/b, 2/a/b), (2/a/b, \mathbf{1}/a/b, 2/a/b)\}$. Let us note that in order to extrapolate the probability of correctness of each instanced evolution, the function $P_{\mathcal{O},\mathcal{F}}$ always takes the observation providing the highest value (see Definition 7). According to $\mathcal{O}$, this means that the probability of an instanced evolution $(t_1/a/b, t_2/a/b, t_3/a/b)$, with $t_2 \geq 1.5$, will be computed from the first observation, while the second observation will be used if $t_2 < 1.5$. Thus, we have that the expression

$$\int_0^{t'}\int_0^5\int_0^4 P_{\mathcal{O},\mathcal{F}}(t_1/a/b, t_2/a/b, t_3/a/b)$$

where $t' = \texttt{min}\{3, (10 - (t_1 + t_2))\}$, is equal to

$$\begin{aligned}
&\int_0^{t'}\int_0^{\mathbf{1.5}}\int_0^4 \left(\frac{1}{2} - |2 - t_1| + 1\right) \cdot \\
&\qquad \left(\frac{2}{3} - |\mathbf{1} - t_2| + 1\right) \cdot \\
&\qquad \left(\frac{3}{4} - |2 - t_3| + 1\right) \quad dt_1\, dt_2\, dt_3 \\
&\qquad\qquad + \\
&\int_0^{t'}\int_{\mathbf{1.5}}^5\int_0^4 \left(\frac{1}{2} - |2 - t_1| + 1\right) \cdot \\
&\qquad \left(\frac{2}{3} - |\mathbf{2} - t_2| + 1\right) \cdot \\
&\qquad \left(\frac{3}{4} - |2 - t_3| + 1\right) \quad dt_1\, dt_2\, dt_3
\end{aligned}$$

□

Once we are provided with a method to add the probability of correctness of all instanced evolutions associated to an evolution, the coverage measure of a set of observations/tests will be the addition of these values for all the available evolutions. Actually, we will only check those evolutions that can be produced by the specification, have n inputs, and last at most t time units. For each of them, we will add the probability that each instanced evolution fitting into the evolution works properly in the SUT (which is calculated by the CP function). Since only sequences of n inputs that can be produced in at most t time units are considered, the set of these evolutions is finite (let us note that it could be infinite if one of these conditions were missing). In the next definition, the set of evolutions to be considered is denoted by E.

Definition 10 Let $Spec$ and SUT be TFSMs, $n \in \mathbb{N}$ and $t \in \mathbb{R}_+$. Let

$$E = \left\{ e \;\middle|\; \begin{array}{l} e = ([t_{11}, t_{12})/i_1/o_1, \ldots, [t_{n1}, t_{n2})/i_n/o_n) \\ \wedge\, e \in \texttt{TEvol}(Spec) \;\wedge\; \sum_i t_{i2} \leq t \end{array} \right\}$$

Besides, let $\mathcal{O} \subseteq \texttt{InsTEvol}(SUT)$ be a set of instanced evolutions of SUT and $\mathcal{F}$ be a a probabilistic stability function for SUT. The *coverage* of $\mathcal{O}$, denoted by $\texttt{coverage}(\mathcal{O})$, is given by the expression $\sum_{e \in E} CP_{\mathcal{O},\mathcal{F}}(e)$. □

The previous metric allows to compare observations as well as their direct counterpart, tests. A test suite is better than another if the observations we would collect from its application (which are unique up to *correctness*) provide a better coverage. This is so because if the test suite is applied to the SUT and all of them are passed (i.e., if the expected observations are obtained), then we will have a higher confidence on the correctness of the SUT. So, test suites can be compared by considering their corresponding observations. Moreover, we can define the *optimal test suite* fitting into a given maximal size. Instead of considering just the number of tests, the size of a test suite will be the addition of the length of each test in the suite, that is, the number of inputs we wish to offer among all tests included in the suite. Given this maximal size, we may define the best test suite of this size.

Definition 11 Let M be a TFSM and $\mathcal{O}_1, \mathcal{O}_2 \subseteq \texttt{InsTEvol}(M)$. We say that $\mathcal{O}_1$ *provides a better confidence* than $\mathcal{O}_2$, denoted by $\mathcal{O}_1 \sqsupseteq \mathcal{O}_2$, if $\texttt{coverage}(\mathcal{O}_1) \geq \texttt{coverage}(\mathcal{O}_2)$.

Let $N \in \mathbb{N}$. We have that the set of instanced evolutions $\mathcal{O} \subseteq \texttt{InsTEvol}(M)$ is *optimal* for the size N if $\sum\{\!|k|(t_1/i_1/o_1, \ldots, t_k/i_k/o_k) \in \mathcal{O}|\!\} \leq N$ and for all $\mathcal{O}' \subseteq \texttt{InsTEvol}(M)$ such that $\sum\{\!|k|(t_1/i_1/o_1, \ldots, t_k/i_k/o_k) \in \mathcal{O}'|\!\} \leq N$ we have $\mathcal{O} \sqsupseteq \mathcal{O}'$. □

Once we choose a set of observations $\mathcal{O}$ providing a good coverage, it is easy to construct from it a test suite such that, if all tests are passed by the SUT, then the set of sequences observed during the application of the tests is $\mathcal{O}$. In this way, a test suite providing good coverage will be constructed. For each observation $ie \in \mathcal{O}$ we will construct a test T such that, after offering the sequence of inputs included in ie at the times specified by it, only the sequence of outputs of ie is accepted. In this case, we will say that T is the *checker* of ie. We will derive a test suite consisting of the checkers of all instanced evolutions in $\mathcal{O}$.

Definition 12 Let M be a TFSM. Given an instanced evolution $ie = (\bar{t}, \sigma) = (t_1/i_1/o_1, \ldots, t_n/i_n/o_n) \in \texttt{InsTEvol}(M)$, the *checker* of ie, denoted by $\texttt{checker}(ie)$, is a test T such that

- $T \xrightarrow{\sigma}_{\bar{t}} s^T$ with $S_P = \{s^T\}$,
- for all instanced evolution $ie' = (\overline{t'}, \sigma') = (t_1/i_1/o_1, \ldots, t_k/i_k/o'_k)$, such that $k \leq n$ and $o'_k \neq o_k$, we have $T \xrightarrow{\sigma'}_{\overline{t'}} s^T$ with $s^T \in S_F$.
- there does not exist $ie'' = (\overline{t''}, \sigma'')$, with $\texttt{len}(ie'') > \texttt{len}(ie)$, such that $T \xrightarrow{\sigma''}_{\overline{t''}} s^T$ for any s^T.

Given $O \subseteq$ $\texttt{InsTEvol}(M)$, we say that the set $\{\texttt{checker}(ie) | ie \in \mathcal{O}\}$ is the *test suite derived from* $\mathcal{O}$ and we denote it by $\texttt{derived}(\mathcal{O})$. $\square$

Lemma 1 Let M be a $\texttt{TFSM}$, $ie \in \texttt{InsTEvol}(M)$, and $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, D)$ be a test such that $T = \texttt{checker}(ie)$. Let us suppose that $M \| T \xrightarrow{\sigma}_{\bar{t}} s^T$. We have $s^T \in S_P$ iff $ie = (\bar{t}, \sigma)$.

4 Conclusions

Testing real-time systems is an intrinsically difficult task because, in general, it is not possible to rigorously test a SUT, neither completely nor in the *limit* (i.e., up to sequences with n inputs produced in less than t time units, for some n and t given). Unfortunately, in practice we can only apply a tiny part of the infinite tests we should apply to claim with certainty the correctness of the SUT. Since it is impossible to fight against this limitation, we consider that one of the goals of a testing methodology is to measure how much information is lost by taking some tests instead of others. In this paper we have seen that, by assuming some simple information about the SUT (which does not directly reveal information about the correctness of any functionality), it is possible to measure how much information is lost by taking some tests or, considering it the other way around, it allows to find optimal test suites in each case (in particular, those losing less information). This additional information is an *inertia model* of the SUT, that is, a function denoting in probabilistic terms the tendency of the SUT to change its state.

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

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

[3] L. Brandán Briones and E. Brinksma. Testing real-time multi input-output systems. In *7th Int. Conf. on Formal Engineering Methods, ICFEM'05, LNCS 3785*, pages 264–279. Springer, 2005.

[4] R. Cardell-Oliver. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.

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

[6] K. Derderian, R. Hierons, M. Harman, and Q. Guo. Automated Unique Input Output sequence generation for conformance testing of FSMs. *The Computer Journal*, 49(3):331–344, 2006.

[7] A. En-Nouaary and R. Dssouli. A guided method for testing timed input output automata. In *15th Int. Conf. on Testing Communicating Systems, TestCom'03, LNCS 2644*, pages 211–225. Springer, 2003.

[8] M. Fecko, M. Uyar, A. Duale, and P. Amer. A technique to generate feasible tests for communications systems with multiple timers. *IEEE/ACM Transactions on Networking*, 11(5):796–809, 2003.

[9] Q. Guo, R. Hierons, M. Harman, and K. Derderian. Computing Unique Input/Output sequences using genetic algorithms. In *3rd Int. Workshop on Formal Approaches to Testing of Software, FATES'03, LNCS 2931*, pages 164–177. Springer, 2003.

[10] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.

[11] T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating test cases for a timed I/O automaton model. In *12th Int. Workshop on Testing of Communicating Systems, IWTCS'99*, pages 197–214. Kluwer Academic Publishers, 1999.

[12] G.-D. Huang and F. Wang. Automatic test case generation with region-related coverage annotations for real-time systems. In *3rd Int. Symposium on Automated Technology for Verification and Analysis, ATVA'05, LNCS 3707*, pages 144–158. Springer, 2005.

[13] M. Krichen and S. Tripakis. An expressive and implementable formal framework for testing real-time systems. In *17th Int. Conf. on Testing of Communicating Systems, TestCom'05, LNCS 3502*, pages 209–225. Springer, 2005.

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

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

[16] M. Núñez and I. Rodríguez. Encoding `PAMR` into (timed) `EFSMs`. In *22nd IFIP WG 6.1 Int. Conf. on Formal Methods for Networked and Distributed Systems, FORTE'02, LNCS 2529*, pages 1–16. Springer, 2002.

[17] M. Núñez and I. Rodríguez. Conformance testing relations for timed systems. In *5th Int. Workshop on Formal Approaches to Software Testing, FATES'05, LNCS 3997*, pages 103–117. Springer, 2006.

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

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

[20] J. Springintveld, F. Vaandrager, and P. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001. Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.

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