Testing finite state machines presenting stochastic time and timeouts ⋆

Mercedes G. Merayo, Manuel Núñez and Ismael Rodríguez

Dept. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, 28040 Madrid, Spain
e-mail: mgmerayo@fdi.ucm.es, {mn,isrodrig}@sip.ucm.es

Abstract. In this paper we define a formal framework to test implementations that can be represented by the class of finite state machines introduced in [21]. In addition to introduce an appropriate notion of test, we provide an algorithm to derive test suites from specifications such that the constructed test suites are sound and complete with respect to two of the conformance relations introduced in [21]. In fact, the current paper together with [21] constitute a complete formal theory to specify and test the class of systems covered by the before mentioned stochastic finite state machines.

1 Introduction

The scale and heterogeneity of current systems make impossible for developers to have an overall view of them. Thus, it is difficult to foresee those errors that are either critical or more probable. In this line, *formal testing techniques* [17,26,5,28] allow to test the correctness of a system with respect to a specification. Formal testing originally targeted the functional behavior of systems, such as determining whether the tested system can, on the one hand, perform certain actions and, on the other hand, does not perform some non-expected ones. While the relevant aspects of some systems only concern *what* they do, in some other systems it is equally relevant *how* they do what they do. Thus, after the initial consolidation stage, formal testing techniques started also to deal with *non-functional* properties. In fact, there are already several proposals for timed testing (e.g. [20,7,29,9,23,24,16,14,3,25]). In these papers, with the only exception of [24], time is considered to be *deterministic*, that is, time requirements follow the form "after/before t time units..." In fact, in most of the cases time is introduced by means of clocks following [1]. Even though the inclusion of time allows to give a more precise description of the system to be implemented, there are frequent situations that cannot be accurately described by using this notion of deterministic time. For example, we may desire to specify a system where a message is

⋆ Research partially supported by the Spanish MEC projects MASTER/TERMAS (TIC2003-07848-C02-01) and WEST/FAST (TIN2006-15578-C02-01), and the Marie Curie project TAROT (MRTN-CT-2003-505121).

expected to be received with probability $\frac{1}{2}$ in the interval $(0, 1]$, with probability $\frac{1}{4}$ in $(1, 2]$, and so on.

In order to use a formal technique, we need that the systems under study can be expressed in terms of a formal language. While the first languages represented only the functional behavior of systems, a new generation of languages allowed to explicitly represent non-functional aspects of systems such as the probability of performing a certain task [15,10,30,6,19] or the time consumed by the system while performing tasks, being it either given by fix amounts of time [27,22,11] or defined in probabilistic/stochastic terms [13,2,12,18,4,8]. A suitable representation of the temporal behavior is critical for constructing useful models of real-time systems. A language to represent these systems should enable the definition of temporal conditions that may direct the system behavior, as well as the time consumed by the execution of tasks. In this line, the time consumed during the execution of a system falls into one of the following categories:

(a) The system consumes time while it performs its operations. This time may depend on the values of certain parameters of the system, such as the available resources.
(b) The time passes while the system waits for a reaction from the environment. In particular, the system can change its internal state if an interaction is not received before a certain amount of time.

A language focussing on temporal issues should allow the specifier to define how the system behavior is affected by both kinds of temporal aspects (e.g., a task is performed if executing the previous task took too much time, if the environment does not react for a long time, if the addition of both times exceeds a given threshold, etc). Even though there exists a myriad of timed extensions of classical frameworks, most of them specialize only in one of the previous variants: Time is either associated with actions or associated with delays/timeouts. In this paper we use the formalism introduced in [21] that allows to specify in a natural way both time aspects. In our framework, timeouts are specified by using fix amounts of time. In contrast, the duration of actions will be given by *random variables*. That is, instead of having expressions such as "the action o takes t units of time to be performed" we will have expressions such as "with probability p the action o will be performed before t units of time". We will consider a suitable extension of finite state machines where (stochastic) time information is included. Intuitively, transitions in finite state machines indicate 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'. An appropriate notation for such a transition could be $s \xrightarrow{i/o} s'$. If we consider a timed extension of finite state machines, transitions as $s \xrightarrow{i/o}_t s'$ indicate that the time between receiving the input i and returning the output o is equal to t. In the new model that we introduce in this paper for stochastic transitions, we will consider that the time consumed between the input is applied and the output is received is given by a random variable ξ. Thus the interpretation of a transition $s \xrightarrow{i/o}_\xi s'$ is "if the machine is in state s and receives an input i then it will produce the output

o before time t with probability $P(\xi \leq t)$ and it will change its state to s'". The definition of conformance testing relations is more difficult than usually. In particular, even in the absence of non-determinism, the same sequence of actions may take different time values to be performed in different runs of the system. While the definition of the new language is not difficult, mixing these temporal requirements strongly complicates the posterior theoretical analysis.

As we have already indicated, this paper represents a continuation of the work initiated in [21]. In that paper we proposed several *stochastic-temporal conformance relations*: An implementation is correct with respect to a specification if it does not show any behavior that is forbidden by the specification, where both the functional behavior and the temporal behavior are considered (and, implicitly, how they affect each other). In this paper we introduce a notion of test and how to test implementations that can be represented by using our notion of finite state machine. In addition, we provide an algorithm that derives test suites from specifications. The main result of our paper indicates that these test suites have the same distinguishing power as the two most interesting conformance relations presented in [21] in the sense that an implementation successfully passes a test suite iff it is conforming to the specification.

Let us note that testing the stochastic behavior of a system affected by non-determinism requires to face some issues that are not considered by other testing approaches. In particular, contrary to usual approaches, providing an *incorrectness* diagnosis may require to consider the result of *all* tests in a test suite, because a single test result could be insufficient to claim the incorrectness of the IUT. Regarding temporal performance requirements, our testing methodology will take into account that the system is only responsible for the (a) type consumed time, not for that of (b) type. That is, we have to distinguish between time associated with the performance of tasks and passing of time due to the possible inactivity of the operator of the system. In addition, testing this kind of stochastic systems present some specific difficulties. In fact, since the implementations are treated as black boxes, we have no access to the random variables governing its behavior. Thus, we cannot determine whether they appropriately correspond to the random variables appearing in the specification. We will use an approach introduced in [24] consisting in collecting different time values from the application of tests to the implementation and performing a *hypothesis contrast* to determine where these observed execution times *fit* the random variable indicated by the specification.

The rest of the paper is structured as follows. In the next two sections we remind our notion of stochastic finite state machine and the two most interesting implementation relations introduced in [21]. In Section 4 we formally define a notion of test, as well as the application of tests to implementations and two notions of successfully passing a test suite. In Section 5 we present an algorithm to derive test suites and show that the derived test suites appropriately capture the relations introduced in Section 3. Finally, in Section 6 we present our conclusions. In addition, the paper contains two appendixes, presented only for the reviewers and non being integral parts of the paper. The fist appendix includes the proof of the main result of the paper while the second one shows how hypothesis contrasts can be performed (this appendix can be found in [21]).

2 A stochastic extension of the `EFSM` model

In this section we introduce our notion of finite state machines with stochastic time. We use random variables to model the (stochastic) time output actions take to be executed. Thus, we need to introduce some basic concepts on random variables. We will consider that the sample space, that is, the domain of random variables, is a set of numeric time values `Time`. Since this is a *generic* time domain, the specifier can choose whether the system will use a discrete/continuous time domain. We simply assume that $0 \in \texttt{Time}$. Regarding passing of time, we will also consider that machines can evolve 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.

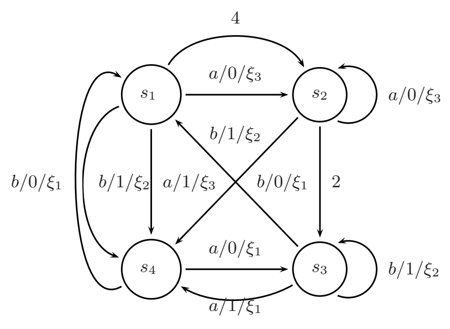
During the rest of the paper we will use the following notation. Tuples of elements $(e_1, e_2 \dots, e_n)$ will be denoted by $\bar{e}$. $\hat{a}$ denotes an interval of elements $[a_1, a_2)$, with $a_1, a_2 \in \texttt{Time}$ and $a_1 < a_2$. We will use the projection function π_i such that given a tuple $\bar{t} = (t_1, \dots, t_n)$, for all $1 \leq i \leq n$ we have $\pi_i(\bar{t}) = t_i$. Let $\bar{t} = (t_1, \dots, t_n)$ and $\bar{t}' = (t'_1, \dots, t'_n)$. We denote by $\sum \bar{t}$ the addition of all the elements belonging to the tuple $\bar{t}$, that is, $\sum_{j=1}^{n} t_j$. The number of elements of the tuple will be represented by $|\bar{t}|$. Finally, if $\bar{t} = (t_1 \dots t_n)$, $\bar{p} = (\hat{t_1} \dots \hat{t_n})$ and for all $1 \leq j \leq n$ we have $t_j \in \hat{t_j}$, we write $\bar{t} \in \bar{p}$.

Definition 1. We denote by $\mathcal{V}$ the set of random variables ($\xi, \psi, \dots$ range over $\mathcal{V}$). Let ξ be a random variable. We define its *probability distribution function* as the function $F_\xi : \texttt{Time} \longrightarrow [0,1]$ such that $F_\xi(x) = P(\xi \leq x)$, where $P(\xi \leq x)$ is the probability that ξ assumes values less than or equal to x.

Given two random variables ξ and ψ we consider that $\xi + \psi$ denotes a random variable distributed as the addition of the two random variables ξ and ψ. We will call *sample* to any multiset of elements belonging to `Time`. We denote the set of multisets in `Time` by $\wp(\texttt{Time})$. Let ξ be a random variable and J be a sample. We denote by $\gamma(\xi, J)$ the *confidence* of ξ on J. $\square$

In the previous definition, a sample simply contains an observation of values. In our setting, samples will be associated with the time values that implementations take to perform sequences of actions. We have that $\gamma(\xi, J)$ takes values in the interval $[0, 1]$. Intuitively, bigger values of $\gamma(\xi, J)$ indicate that the observed sample J is more likely to be produced by the random variable ξ. That is, this function decides how *similar* the probability distribution function generated by J and the one corresponding to the random variable ξ are. In the appendix of this paper we show one of the possibilities to formally define the notion of confidence by means of a hypothesis contrast.

Definition 2. A *Stochastic Finite State Machine*, in short `SFSM`, is a tuple $M = (S, I, O, \delta, TO, s_{in})$ where S is the set of states, with $s_{in} \in S$ being the



$$F_{\xi_1}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ \frac{x}{5} & \text{if } 0 < x < 5 \\ 1 & \text{if } x \geq 5 \end{cases}$$

$$F_{\xi_2}(x) = \begin{cases} 0 & \text{if } x < 4 \\ 1 & \text{if } x \geq 4 \end{cases}$$

$$F_{\xi_3}(x) = \begin{cases} 1 - e^{-2\cdot x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Fig. 1. Example of Stochastic Finite State Machine.

initial state, I and O denote the sets of input and output actions, respectively, δ is the set of transitions, and $TO : S \longrightarrow S \times (\texttt{Time} \cup \{\infty\})$ is the *timeout function.* Each transition belonging to δ is a tuple (s, i, o, ξ, s') where $s, s' \in S$ are the initial and final states, $i \in I$ and $o \in O$ are the input and output actions, and $\xi \in \mathcal{V}$ is the random variable defining the time associated with the transition.

Let $M = (S, I, O, \delta, TO, s_{in})$ be a `SFSM`. We say that M is *input-enabled* if for all state $s \in S$ and input $i \in I$ there exist s', o, ξ, such that $(s, i, o, \xi, s') \in \delta$. We say that M is *deterministically observable* if for all s, i, o there do not exist two different transitions $(s, i, o, \xi_1, s_1), (s, i, o, \xi_2, s_2) \in \delta$. □

Intuitively, a transition (s, i, o, ξ, s') indicates that if the machine is in state s and receives the input i then the machine emits the output o before time t with probability $F_\xi(t)$ and the machine changes its current state to s'.

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

Example 1. Let us consider the machine depicted in Figure 1 in which the initial state is s_1 (i.e. $s_0 = s_1$). Each transition has an associated random variable. In the following we explain how the random variables are distributed. Let us consider that ξ_1 is *uniformly distributed* in the interval $[0, 5]$. Uniform distributions assign equal probability to all the times in the interval. The random variable ξ_2 follows a Dirac distribution in 4. The idea is that the corresponding delay will be equal to 4 time units. Finally, ξ_3 is *exponentially* distributed with parameter 2. Let us consider the transition $(s_1, (b, 1, \xi_2), s_4)$. Intuitively, if the machine is in state s_1 and it receives the input b then it will produce the output 1 after a time given by ξ_2. For example, we know that this time will be less than 1 time unit with probability $\frac{1}{5}$, it will be less than 3 time units with probability $\frac{3}{5}$, and so on. Finally, once 5 time units have passed we know that the output 1 has been performed (that is, we have probability 1). Regarding the timeout function we have $TO(s_1) = (s_2, 4)$. In this case, if the machine is in state s_1 and no input is received before 4 units of time then the state is changed to s_2.

Definition 3. Let $M = (S, I, O, \delta, TO, s_{in})$ be a `SFSM`. We say that a tuple $(s_0, s, i/o, \hat{t}, \xi)$ is a *step* for the state s_0 of M if there exist k states $s_1, \ldots, s_k \in S$, with $k \geq 0$, such that $\hat{t} = \left[\sum_{j=0}^{k-1} \pi_2(TO(s_j)), \sum_{j=0}^{k} \pi_2(TO(s_j))\right)$ and there exists a transition $(s_k, i, o, \xi, s) \in \delta$.

We say that $(\hat{t}_1/i_1/\xi_1/o_1, \ldots, \hat{t}_r/i_r/\xi_r/o_r)$ is a *stochastic evolution* of M if there exist r steps of M $(s_{in}, s_1, i_1/o_1, \hat{t}_1, \xi_1), \ldots, (s_{r-1}, s_r, i_r/o_r, \hat{t}_r, \xi_r)$ for the states $s_{in} \ldots s_{r-1}$, respectively. We denote by $\texttt{SEvol}(M)$ the set of stochastic evolutions of M. In addition, we say that $(\hat{t}_1/i_1/o_1, \ldots, \hat{t}_r/i_r/o_r)$ is a *functional evolution* of M. We denote by $\texttt{FEvol}(M)$ the set of functional evolutions of M. We will use the shortenings $(\sigma, \bar{p})$ and $(\sigma, \bar{p}, \bar{\xi})$ to denote a functional and a stochastic evolution, respectively, where $\sigma = (i_1/o_1 \ldots i_r/o_r)$, $\bar{p} = (\hat{t_1} \ldots \hat{t_r})$ and $\bar{\xi} = (\xi_1 \ldots \xi_r)$. □

Intuitively, a step is a sequence of transitions that contains an action transition preceded by zero or more timeouts. The interval $\hat{t}$ indicates the time values where the transition could be performed. In particular, if the sequence of timeouts is empty then we have the interval $\hat{t} = [0, TO(s_0))$. An evolution is a sequence of inputs/outputs corresponding to the transitions of a chain of steps, where the first one begins with the initial state of the machine. In addition, stochastic evolutions also include time information which inform us about possible timeouts (indicated by the intervals $\hat{t}_j$) and random variables associated to the execution of each output after receiving each input in each step of the evolution. In the following definition we introduce the concept of *instanced evolution*. Intuitively, instanced evolutions are constructed from evolutions by instantiating to a concrete value each timeout, given by an interval, of the evolution.

Definition 4. Let $M = (S, I, O, \delta, TO, s_{in})$ be a `SFSM` and let us consider a *stochastic evolution* $e = (\hat{t}_1/i_1/\xi_1/o_1, \ldots, \hat{t}_r/i_r/\xi_r/o_r)$. We say that the tuple $(t_1/i_1/\xi_1/o_1, \ldots, t_r/i_r/\xi_r/o_r)$ is an *instanced stochastic evolution of* e if for all $1 \leq j \leq r$ we have $t_j \in \hat{t}_j$. Besides, we say that the tuple $(t_1/i_1/o_1, \ldots, t_r/i_r/o_r)$ is an *instanced functional evolution* of e.

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

3 Implementation relations

In this section we remind two of the implemenation relations introduced in [21]. First, we give an implementation relation to deal with functional aspects. It follows the pattern borrowed from $\texttt{conf}_{nt}$ [23]: An implementation I *conforms* to a specification S if for all possible evolution of S the outputs that the implementation I may perform after a given input are a subset of those for the specification.

In addition to the non-stochastic conformance of the implementation, we require other additional conditions, related to time, to hold. Specifically, we require that the implementation always complies with the timeouts established by the specification.

A specification is a stochastic finite state machine. Regarding implementations, we consider that they are also given by means of SFSMs. We will consider that both specifications and implementations are given by deterministically observable SFSMs. Besides, we assume that input actions are always enabled in any state of the implementation, that is, implementations are input-enabled according to Definition 2. This is a usual condition to assure that the implementation will react (somehow) to any input appearing in the specification. First, we introduce the implementation relation $\texttt{conf}_f$, where only functional aspects of the system (i.e., which outputs are allowed/forbidden and how timeouts are defined) are considered while the performance of the system (i.e., how fast outputs are executed) is ignored. Let us note that the time spent by a system waiting for the environment to react has the capability of affecting the set of available outputs of the system. This is because this time may trigger a change of the state. So, a relation focusing on functional aspects must explicitly take into account the maximal time the system may stay in each state. This time is given by the *timeout* of each state.

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

Intuitively, the idea underlying the definition of the functional conformance relation $I \; \texttt{conf}_f \; S$ is that the implementation I does not *invent* anything for those sequences of inputs that are *specified* in the specification S. Let us note that if the specification has also the property of input-enabled then we may remove the condition "for each functional evolution $e \in \texttt{FEvol}(S)$, with $e = (\hat{t}_{t1}/i_1/o_1, \ldots, \hat{t}_{tr}/i_r/o_r)$ and $r \geq 1$".

In addition to requiring this notion of *functional* conformance, we have to ask for some conditions on delays. A first approach would be to require that the random variables associated with evolutions of the implementation are identically distributed as the ones corresponding to the specification. However, the fact that we assume a black-box testing framework disallows us to check whether these random variables are indeed identically distributed. Thus, we have to give more *realistic* implementation relations based on finite sets of observations. Next, we present two implementation relations that are less *accurate* but that are *checkable*. These relations take into account the observations that we may get from the implementation. We will collect a sample of time values and we will *compare* this sample with the random variables appearing in the specification. By comparison we mean that we will apply a contrast to decide, with a certain confidence, whether the sample could be generated by the corresponding random variable.

Definition 6. Let I be a $\texttt{SFSM}$. We say that $(\sigma, \bar{t}, \bar{t}')$, with $\sigma = i_1/o_1, \ldots, i_n/o_n$, $\bar{t} = (t_1 \ldots t_n)$, and $\bar{t}' = (t'_1 \ldots t'_n)$, is an *observed time execution* of I, or simply *time execution*, if the observation of I shows that for all $1 \leq j \leq n$ we have that the time elapsed between the acceptance of the input i_j and the observation of the output o_j is t'_j units of time, being the input i_j accepted t_j units of time after the last output was observed.

Let $\Phi = \{(\sigma_1, \bar{t}_1), \ldots, (\sigma_m, \bar{t}_m)\}$t and let $H = \{\!|(\sigma'_1, \bar{t}_{d1}, \bar{t}_{o1}), \ldots, (\sigma'_n, \bar{t}_{dn}, \bar{t}_{on})|\!\}$ be a multiset of timed executions. We say that $\texttt{Sampling}^k_{(H,\Phi)} : \Phi \longrightarrow \wp(\texttt{Time})$ is a *k-sampling application* of H for Φ if $\texttt{Sampling}^k_{(H,\Phi)}(\sigma, \bar{t}) = \{\!|\pi_k(\bar{t}_o) \mid (\sigma, \bar{t}, \bar{t}_o) \in H \ \wedge\ |\sigma| \geq k|\!\}$, for all $(\sigma, \bar{t}) \in \Phi$. We say that $\texttt{Sampling}_{(H,\Phi)} : \Phi \longrightarrow \wp(\texttt{Time})$ is a *sampling application* of H for Φ if $\texttt{Sampling}_{(H,\Phi)}(\sigma, \bar{t})) = \{\!|\sum \bar{t}_o \mid (\sigma, \bar{t}, \bar{t}_o) \in H|\!\}$, for all $(\sigma, \bar{t}) \in \Phi$. $\square$

Regarding the definition of k-sampling applications, we just associate with each subtrace of length k the observed time of each transition of the execution at length k. In the definition of sampling applications, we assign to each trace the total observed time corresponding to the whole execution.

Definition 7. Let I and S be $\texttt{SFSMs}$, H be a multiset of timed executions of I, $0 \leq \alpha \leq 1$, $\Phi = \{(\sigma, \bar{t}) \mid \exists\, \bar{t}_o : (\sigma, \bar{t}, \bar{t}_o) \in H\} \cap \texttt{InsFEvol}(S)$, and let us consider $\texttt{Sampling}_{(H,\Phi)}$ and $\texttt{Sampling}^k_{(H,\Phi)}$, for all $1 \leq k \leq max\{|\sigma| \mid (\sigma, \bar{t}) \in \Phi\}$.

We say that I $(\alpha, H)-$*weak stochastically conforms* to S, and we denote it by $I\ \texttt{confs}^{(\alpha,H)}_w\ S$, if $I\ \texttt{conf}_f\ S$ and for all $(\sigma, \bar{t}) \in \Phi$ we have

$$\begin{array}{c}(\sigma, \bar{t}, \bar{\xi}) \in \texttt{InsSEvol}(S) \\ \Downarrow \\ \gamma\left(\sum \bar{\xi}, \texttt{Sampling}_{(H,\Phi)}(\sigma, \bar{t})\right) > \alpha\end{array}$$

We say that I $(\alpha, H)-$*strong stochastically conforms* to S, and we denote it by $I\ \texttt{confs}^{(\alpha,H)}_s\ S$, if $I\ \texttt{conf}_f\ S$ and for all $(\sigma, \bar{t}) \in \Phi$ we have

$$\begin{array}{c}(\sigma, \bar{t}, \bar{\xi}) \in \texttt{InsSEvol}(S) \\ \Downarrow \\ \forall\, 1 \leq j \leq |\sigma| : \gamma(\pi_j(\bar{\xi}), \texttt{Sampling}^j_{(H,\Phi)}(\sigma, \bar{t})) > \alpha\end{array}$$

$\square$

The idea underlying the new relations is that the implementation must conform to the specification in the usual way (that is, $I\ \texttt{conf}_f\ S$). Besides, for all observation of the implementation that can be performed by the specification, the observed execution time values *fit* the random variable indicated by the specification. This notion of *fitting* is given by the function γ that it is formally defined in the Appendix 2 of this paper. While the *weak* notion only compares the total time, the *strong* notion checks that the time values are appropiate for each performed output.

4 Tests cases for stochastic systems

We consider that tests represent sequences of inputs applied to an IUT. 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 I/O labelled transition systems (i.e. trees) with a strict alternation between an input action and the set of output actions. After an output action we may find either a leaf or another input action. Leaves can be labelled either by *pass* or by *fail*. In addition to check the functional behavior of the IUT, test have also to detect whether wrong timed behaviors appear. Thus, tests have to include capabilities to deal with the two ways of specifying time. On the one hand, we will include *random variable*. The idea is that we will record the time that it takes for the implementation to arrive to that point. We will collect a sample of times (one for each test execution) and we will *compare* this sample with the random variable. By comparison we mean that we will apply a contrast to decide, with a certain confidence, whether the sample could be generated by the corresponding random variable. On the second hand, tests will include *delays* before offering input actions. The idea is that delays in tests will induce timeouts in IUTs. Thus, we may indirectly check whether the timeouts imposed by the specification are reflected in the IUT by offering input actions after a specific delay.

Definition 8. A *test case* is a tuple $T = (S, I, O, \lambda, s_0, S_I, S_O, S_F, S_P, \zeta, D)$ where S is the set of states, I and O, with $I \cap O = \emptyset$ are the sets of input and output actions, respectively, $\lambda \subseteq S \times I \cup O \times S$ 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 $(s, a, s') \in \lambda$. 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 $(s, o, s') \in \lambda$. In this case, $s' \notin S_O$. Moreover, there do not exist $i \in I, s' \in S$ such that $(s, i, s') \in \lambda$.
- 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 $(s, a, s') \in \lambda$.

Finally, we have two timed functions. $\zeta : S_P \longrightarrow \bigcup_{j=1}^{\infty} \mathcal{V}^j$ is a function associating random variables, to compare with the time that took to perform the outputs, with passing states. $D : S_I \longrightarrow \texttt{Time}$ is a function associating delays with input states.

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

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

Let T be a valid test, $\sigma = i_1/o_1, \dots, i_r/o_r$, s^T be a state of T, and $\bar{t} = (t_1, \dots, t_r) \in \mathtt{Time}^r$. We write $T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ if $T \overset{\sigma}{\Longrightarrow} s^T$, $t_1 = D(s_0)$, and for all $1 < j \leq r$ we have $t_j = D(s_{j1})$.

□

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

Definition 9. Let I be $\mathtt{SFSM}$ and $T = (S_t, I, O, \delta_T, s_0, S_I, S_O, S_F, S_P, \zeta, D)$ be a valid test, $\sigma = i_1/o_1, \dots, i_r/o_r$, s^T be a state of T, $\bar{t} = (t_1, \dots, t_r)$, and $\bar{t}_o = (t_{o1}, \dots, t_{or})$. We write $I \| T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ if $T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ and $(\sigma, \bar{t}) \in \mathtt{InsFEvol}(I)$. We write $I \parallel T \overset{\sigma}{\Longrightarrow}_{\bar{t}, \bar{t_o}} s^T$ if $I \parallel T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ and $(\sigma, \bar{t}, \bar{t}_o)$ is a observed timed execution of I. In this case we say that $(\sigma, \bar{t}, \bar{t}_o)$ is a *test execution* of I and T.

We say that I *passes* the test suite $\mathcal{T}_{st}$, denoted by $\mathtt{pass}(I, \mathcal{T}_{st})$, if for all test $T \in \mathcal{T}_{st}$ there do not exist $(\sigma, \bar{t}) \in \mathtt{InsFEvol}(I)$, $s^T \in S$ such that $I \parallel T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ and $s^T \in S_F$. □

Let us remark that since we are assuming that implementations are input-enabled, the testing process will conclude only when the test reaches either a fail or a success state.

In addition to this notion of passing tests, we will have different time conditions. We apply the time conditions to the set of *observed timed executions*, not to stochastic evolutions of the implementations, due to the fact that stochastic evolutions do not have a single time value that we can directly compare with the time stamp attached to the pass state. In fact, we need a set of test executions associated to each evolution to evaluate if they match the distribution function associated to the random variable indicated by the corresponding state of the test. In order to increase the degree of reliability, we will not take the classical approach where passing a test suite is defined according only to the results for each test. In our approach, we will put together all the observations, for each test, so that we have more samples for each evolution. In particular, some observations will be used several times. In other words, an observation from a given test may be used to check the validity of another test sharing the same observed sequence.

Definition 10. Let I be a $\mathtt{SFSM}$ and $\mathcal{T}_{st} = \{T_1, \dots, T_n\}$ be a test suite. Let $H_1, \dots, H_n$ be test execution samples of I and $T_1, \dots, T_n$, respectively. Let $H = \bigcup_{i=1}^{n} H_i$, $\varPhi = \{(\sigma, \bar{t}) \mid \exists\ \bar{t}_o : (\sigma, \bar{t}, \bar{t}_o) \in H\}$, $0 \leq \alpha \leq 1$ and let us consider $\mathtt{Sampling}_{(H,\varPhi)}$ and $\mathtt{Sampling}^k_{(H,\varPhi)}$, for all $1 \leq k \leq max\{|\sigma| \mid (\sigma, \bar{t}) \in \varPhi\}$.

Let $e = (\sigma, \bar{t}) \in \Phi$. We define the set $\texttt{Test}(e, \mathcal{T}_{st}) = \{T \mid T \in \mathcal{T}_{st} \ \wedge \ I \,\|\, T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T\}$.

We say that the implementation I *weakly* $(\alpha, H)-$*passes* the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $e = (\sigma, \bar{t}) \in \Phi$ we have that for all $T \in \texttt{Test}(e, \mathcal{T}_{st})$ such that $I \,\|\, T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ it holds $\gamma(\sum \zeta(s^T), \texttt{Sampling}_{(H,\Phi)}(\sigma, \bar{t})) > \alpha$.

We say that the implementation I *strongly* $(\alpha, H)-$*passes* the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $e = (\sigma, \bar{t}) \in \Phi$ we have that for all $T \in \texttt{Test}(e, \mathcal{T}_{st})$ such that $I \,\|\, T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ it holds $\forall\, 1 \leq j \leq |\sigma| : \gamma(\pi_j(\zeta(s^T)), \texttt{Sampling}^j_{(H,\Phi)}(\sigma, \bar{t})) > \alpha$. □

Let us note that an observed timed execution does not return the random variable associated with performing the evolution (that is, the addition of all the random variables corresponding to each transition of the implementation) but the time that it took to perform the evolution. Intuitively, an implementation passes a test if there does not exist an evolution leading to a fail state. Once we know that the functional behavior of the implementation is correct with respect to the test, we need to check time conditions. The set H corresponds to the observations of the (several) applications of the tests belonging to the test suite $\mathcal{T}_{st}$ to I. Thus, we have to decide whether, for each evolution e, the observed time values (that is, $\texttt{Sampling}_{(H,\Phi)}(e)$) *match* the definition of the random variables appearing in the successful state of the tests corresponding to the execution of that evolution (that is, $\zeta(s^T)$). As we commented previously, we assume a function γ, formally defined in the appendix, that can perform this hypothesis contrast.

5 Test derivation: Soundness and completeness

In this section we present an algorithm to derive test cases from specifications and we show that the derived test suites are sound and complete with respect to the two implementation relations presented in Section 3. As usual, the idea underlying our algorithm consists in traversing the specification in order to get all the possible traces in an appropriate way. First, we introduce some additional notation.

Definition 11. Let $M = (S, I, O, \delta, TO, s_{in})$ be a $\texttt{SFSM}$. We consider the following sets:

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

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

$$\texttt{after}(s, i, o, \bar{\xi}) = \begin{cases} (s', (\bar{\xi}, \xi)) & \text{if } \exists\, \xi : (s, i, o, s', \xi) \in \delta \\ \texttt{error} & \text{otherwise} \end{cases}$$

□

```
Input: A specification $M = (S, I, O, \delta, TO, Tr, s_{in})$.
Output: A test case $T = (S', I, O \cup \{\texttt{null}\}, \lambda, s_0, S_I, S_O, S_F, S_P, \zeta, D)$.

Initialization:

– $S' := \{s_0\}, \delta := S_I := S_O := S_F := S_P := \zeta := D := \emptyset$.
– $S_{aux} := \{(s_{in}, null, s_0)\}$.

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

1. If $(s^M, \bar{\xi}, s^T) \in S_{aux}$ then perform the following steps:
   (a) $S_{aux} := S_{aux} - \{(s^M, \bar{\xi}, s^T)\}$.
   (b) $S_P := S_P \cup \{s^T\}$; $\zeta(s^T) := \bar{\xi}$.
2. If $S_{aux} = \{(s^M, \bar{\xi}, s^T)\}$ then perform the following steps :
   (a) Choose $t_d \in \texttt{Time}$
   (b) $S_{aux} := \{(\texttt{afterTO}(s^M, t_d), \bar{\xi}, s^T)\}$
   (c) If $\exists\, i \in I : \texttt{out}(S^M, i) \neq \emptyset$ then perform the following steps:
       i. $S_{aux} := \emptyset$.
       ii. Choose $i$ such that $\texttt{out}(S^M, i) \neq \emptyset$.
       iii. Consider a fresh state $s' \notin S'$ and let $S' := S' \cup \{s'\}$.
       iv. $S_I := S_I \cup \{s^T\}$; $S_O := S_O \cup \{s'\}$; $\lambda := \lambda \cup \{(s^T, i, s')\}$.
       v. $D(s^T) := t_d$.
       vi. For all $o \notin \texttt{out}(S^M, i)$ do {null is in this case}
           – Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
           – $S_F := S_F \cup \{s''\}$; $\lambda := \lambda \cup \{(s', o, s'')\}$.
       vii. For all $o \in \texttt{out}(S^M, i)$ do
           – Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
           – $\lambda := \lambda \cup \{(s', o, s'')\}$.
           – $(s_1^M, \bar{\xi}') := \texttt{after}(s^M, i, o, \bar{\xi})$.
           – $S_{aux} := S_{aux} \cup \{(s_1^M, \bar{\xi}', s'')\}$.
```

Fig. 2. Derivation of test cases from a specification.

The function `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 next function, `afterTO`(s, t) returns the state that would be reached by the system if we start in the state s and t time units elapsed without receiving an input. The last function, `after`$(s, i, o, \bar{\xi})$, computes the *situation* that is reached from a state s after receiving the input i, producing the output o, supposing that $\bar{\xi}$ denotes the random variables associated to the transitions performed. By *situation* we mean a pair denoting the reached state and the tuple of random variables associated to the transitions performed since the system started its performance. Let us also remark that due to the assumption that `SFSM`s are observable we have that `after`$(s, i, o, \bar{\xi})$ is uniquely determined. Besides, we will apply this function only when the side condition holds, that is, we will never receive `error` as result of applying `after`.

The algorithm to derive tests from a specification is given in Figure 2. It is a non-deterministic algorithm that returns a single test. By considering the possible available choices in the algorithm we extract a full test suite from the specification (this set will be infinite in general). For a given specification M, we denote this set of tests by $\texttt{tests}(M)$. Next we explain how the algorithm works. A set of *pending situations* S_{aux} keeps those triplets denoting the possible states and the tuple of random variables that could appear in a state of the test whose definition, that is, its outgoing transitions, has not been completed yet. A triplet $(s^M, \bar{\xi}, s^T) \in S_{aux}$ indicates that we did not complete the state s^T of the test, the tuple of random variables $\bar{\xi}$ associated to the transitions of the specification that have been traversed from the initial state, and the current state in the transversal of the specification is s^M.

Let us consider the steps of the algorithm. The set S_{aux} initially contains a tuple with the initial states (of both the specification and the test) and the initial tuple of random variables (that is, empty tuple of random variables). For each tuple belonging to S_{aux} we may choose one possibility between two choices. It is important to remark that the second choice can be taken only when the set S_{aux} becomes singleton. So, our derived tests correspond to valid tests as given in Definition 8. The first possibility simply indicates that the state of the test becomes a passing state. The second possibility takes an input and generates a transition in the test labelled by this input. At this step, we choose a delay for the next input state. We select a time value and replace the states of the pending situations by the situations that can be reached if we apply as delay for accepting a new input, the time value selected. This is because, during the delay, timeout transitions can be triggered, so changes of states will be prompted by this fact. That fact allow us to consider sequences of timeout transitions, that is, traces where those transitions are triggered because no input action is received by the system.

Then, the whole sets of outputs is considered. If the output is not expected by the implementation (step 2.(c).vi of the algorithm) then a transition leading to a failing state is created. This could be simulated by a single branch in the test, labelled by `else`, leading to a failing state (in the algorithm we suppose that *all* the possible outputs appear in the test). For the expected outputs (step 2.(c).vii of the algorithm) we create a transition with the corresponding output action and add the appropriate tuple to the set S_{aux}.

Finally, let us remark that finite test cases are constructed simply by considering a step where the second inductive case is not applied. Finally, let us comment on the *finiteness* of our algorithm. If we do not impose any restriction on the implementation (e.g., a bound on the number of states) we cannot determine some important information such as the maximal length of the traces that the implementation can perform. In other words, we would need a *coverage criterium* to generate a finite test suite. Since we do not assume, by default, any criteria, all we can do is to say that this is the, in general, infinite test suite that would allow to prove completeness. Obviously, one can impose restrictions such as "generate n tests" or "generate all the tests with m inputs" and *completeness* will be obtained up to that coverage criterium.

Finally, we present the result that relates, for a specification S and an implementation I, implementation relations and application of test suites. The non-timed aspects of our algorithm are based on the algorithm developed for the *ioco* relation. So, in spite of the differences, the non-timed part of the proof of our result is a simple adaptation of that in [31]. Regarding temporal aspects, the result holds because the temporal conditions required to conform to the specification and to pass the test suite are in fact the same. The proof can be found in Appendix 1.

Theorem 1. Let S, I be SFSMs. Let H be test execution samples of I, $0 \leq \alpha \leq 1$, and $\Phi = \{(\sigma, \bar{t}) \mid \exists\, \bar{t}_o : (\sigma, \bar{t}, \bar{t}_o) \in H\} \cap \texttt{InsFEvol}(S)$. We have that:

– $I \; \texttt{confs}_w^{(\alpha,H)} \; S$ iff I *weakly* $(\alpha, H)-$*passes* $\texttt{tests}(S)$.
– $I \; \texttt{confs}_s^{(\alpha,H)} \; S$ iff I *strongly* $(\alpha, H)-$*passes* $\texttt{tests}(S)$.

□

6 Concluding remarks

This paper concludes the work initiated in [21]. There, we presented a new notion of finite state machine to specify, in an easy way, both the passing of time due to timeouts and the time due to the performance of actions. In addition, we presented several implementation relations based on the notion of conformance. These relations shared a common pattern: The implementation must conform to the specification regarding functional aspects. In this paper we introduce a notion of test, how to apply a test suite to an implementation, and what is the meaning of successfully passing a test suite. Even though implementation relations and passing of test suites are, apparently, unrelated concepts, we provide a link between them: We give an algorithm to derive test suites from specifications in such a way that a test suite is successfully passed iff the implementation conforms to the specification. This result, usually known as *soundness* and *completeness*, allows a user that in order to check the correctness of an implementation, it is the same to consider an implementation relation or to apply a derived test suite.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202(1-2):1–54, 1998.
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. M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-Markov processes. *Theoretical Computer Science*, 282(1):5–32, 2002.
5. E. Brinksma and J. Tretmans. Testing transition systems: An annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 187–195. Springer, 2001.
6. D. Cazorla, F. Cuartero, V. Valero, F.L. Pelayo, and J.J. Pardo. Algebraic theory of probabilistic and non-deterministic processes. *Journal of Logic and Algebraic Programming*, 55(1–2):57–103, 2003.
7. D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *3rd Workshop on Object-Oriented Real-Time Dependable Systems, WORDS'97*, pages 199–206. IEEE Computer Society Press, 1997.
8. P.R. D'Argenio and J.-P. Katoen. A theory of stochastic systems part I: Stochastic automata. *Information and Computation*, 203(1):1–38, 2005.
9. A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-method: Testing real time systems. *IEEE Transactions on Software Engineering*, 28(11):1024–1039, 2002.
10. R. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.
11. M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.
12. H. Hermanns. *Interactive Markov Chains*. PhD thesis, Universität Erlangen-Nürnberg, 1998. Also appeared as *LNCS 2428*, Springer, 2002.
13. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
14. 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.
15. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
16. K.G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using Uppaal. In *4th Int. Workshop on Formal Approaches to Testing of Software, FATES'04, LNCS 3395*, pages 79–94. Springer, 2004.
17. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
18. N. López and M. Núñez. A testing theory for generally distributed stochastic processes. In *12th Int. Conf. on Concurrency Theory, CONCUR'01, LNCS 2154*, pages 321–335. Springer, 2001.
19. N. López, M. Núñez, and I. Rodríguez. Specification, testing and implementation relations for symbolic-probabilistic systems. *Theoretical Computer Science*, 353(1–3):228–248, 2006.
20. D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real time systems from logic specifications. *ACM Transactions on Computer Systems*, 13(4):356–398, 1995.
21. M.G. Merayo, M. Núñez, and I. Rodríguez. Implementation relations for stochastic finite state machines. In *3rd European Performance Engineering Workshop, EPEW'06, LNCS 3964*, pages 123–137. Springer, 2006.
22. 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.

23. M. Núñez and I. Rodríguez. Encoding `PAMR` into (timed) `EFSMs`. In *22nd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'02, LNCS 2529*, pages 1–16. Springer, 2002.
24. M. Núñez and I. Rodríguez. Towards testing stochastic timed systems. In *23rd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'03, LNCS 2767*, pages 335–350. Springer, 2003.
25. 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.
26. A. Petrenko. Fault model-driven test derivation from finite state models: Annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 196–205. Springer, 2001.
27. G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.
28. I. Rodríguez, M.G. Merayo, and M. Núñez. $\mathcal{HOTL}$: Hypotheses and observations testing logic. *Journal of Logic and Algebraic Programming (in press)*, 2007. Available at `http://dx.doi.org/10.1016/j.jlap.2007.03.002`.
29. J. Springintveld, F. Vaandrager, and P.R. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001. Previously appeared as Technical Report CTIT-97-17, University of Twente, 1997.
30. M. Stoelinga and F. Vaandrager. A testing scenario for probabilistic automata. In *30th Int. Colloquium on Automata, Languages and Programming, ICALP'03, LNCS 2719*, pages 464–477. Springer, 2003.
31. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.

Appendix 1: Proof of Theorem 1

Proof. We will prove the first result. The technique is similar for the second one.

First, let us show that $I\,\texttt{confs}_w^{(\alpha,H)}\,S$ implies I *weakly* $(\alpha, H)-$*passes* $\texttt{tests}(S)$. We will use the contrapositive, that is, we will suppose that $I\ \texttt{confs}_w^{(\alpha,H)}\ S$ does not hold and we will prove that I does not *weakly* $(\alpha, H)-$*pass* $\texttt{tests}(S)$. If $I\ \texttt{confs}_w^{(\alpha,H)}\ S$ does not hold then we have two possibilities:

– Either $I\ \texttt{conf}_f\ S$ does not hold, or
– there exists $(\sigma, \bar{t}) \in \varPhi$ and $(\sigma, \bar{t}, \bar{\xi}) \in \texttt{InsSEvol}(S)$ such that

$$\gamma(\sum \bar{\xi}, \texttt{Sampling}_{(H,\varPhi)}(\sigma, \bar{t})) \leq \alpha$$

Let us consider the first case, that is, we suppose that $I\ \texttt{conf}_f\ S$ does not hold. Then, there exist $e = (\sigma, \bar{t})$, with $\sigma = (i_1/o_1, \ldots, /i_r/o_r)$ and $\bar{t} = (t_1, \ldots, t_r)$, and $e' = (\sigma', \bar{t})$, with $\sigma' = (i_1/o_1, \ldots, /i_r/o'_r)$, being $r \geq 1$ such that $e \in \texttt{InsFEvol}(S)$, $e' \in \texttt{InsFEvol}(I)$, and $e' \notin \texttt{InsFEvol}(S)$. We will show that there exists a test $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, \zeta, D) \in \texttt{tests}(S)$ such that $T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ with $s^T \in S_P$ and $T \overset{\sigma'}{\Longrightarrow}_{\bar{t}} u^T$ with $u^T \in S_F$. By having such a test T, we obtain $I \parallel T \overset{\sigma'}{\Longrightarrow}_{\bar{t}} u^T$, being u_T a fail state. Thus, we conclude I does not passes $\texttt{tests}(S)$. We will construct this test T by applying the algorithm given in Figure 2, and by resolving the non-deterministic choices in the following way:

1. **for** $1 \leq j \leq r$ **do**
   (a) Apply inductive case 2 for the input action i_j selecting $t_d := t_j$.
   (b) Apply inductive case 1 for all the elements $(s^M, \bar{\xi}, s^T) \in S_{aux}$ that are obtained by processing an output different from o_j.
   endfor
2. Apply first inductive case for the last (i.e. remaining) element $(s^M, \bar{\xi}, s^T) \in S_{aux}$.

Since $e' \notin \texttt{InsFEvol}(S)$, the last application of the second inductive case for the output o'_r must be necessarily associated to the step 2.(c).vi, so the previous algorithm generates a test T such that $T \overset{\sigma'}{\Longrightarrow}_{\bar{t}} u^T$, with $u^T \in S_F$. Then, $I \parallel T \overset{\sigma'}{\Longrightarrow}_{\bar{t},\bar{t}_p} u^T$ for some $\bar{t}_p$. Given the fact that $T \in \texttt{tests}(S)$ we deduce that $\texttt{pass}(I, \texttt{tests}(S))$ does not hold. Thus, we conclude I does not *weakly* $(\alpha, H)-$*pass* $\texttt{tests}(S)$.

Let us suppose now that $I \; \texttt{confs}_w^{(\alpha,H)} \; S$ does not hold due to there exist $(\sigma, \bar{t}) \in \Phi$ and $(\sigma, \bar{t}, \bar{\xi}) \in \texttt{InsSEvol}(S)$ such that

$$\gamma(\sum \bar{\xi}, \texttt{Sampling}_{(H,\Phi)}(\sigma, \bar{t})) \leq \alpha$$

Since $(\sigma, \bar{t}) \in \Phi$ we have that $(\sigma, \bar{t}) \in \texttt{InsFEvol}(S)$. Let us consider the test T built just as we did before. We have again $T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$, with $s^T \in S_P$. The time stamps associated with the state s^T are generated by considering the only tuple of random variables associated to the transitions in which σ could be performed in S considering the delays $\bar{t}$, that is $\zeta(s^T) = \bar{\xi}$. Besides, since $(\sigma, \bar{t}) \in \Phi$ there exists $(\sigma, \bar{t}, \bar{t}_o) \in H$ such that $I \parallel T \overset{\sigma}{\Longrightarrow}_{\bar{t},\bar{t_o}} s^T$. So, we have $T \in \texttt{Test}(e, \mathcal{T}_{st})$ such that $I \parallel T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ and $\gamma(\sum \zeta(s^T), \texttt{Sampling}_{(H,\Phi)}(\sigma, \bar{p})) < \alpha$. We conclude I does not *weakly* $(\alpha, H)-$*pass* $\texttt{tests}(S)$.

Let us prove now that $I \texttt{confs}_w^{(\alpha,H)} S$ implies I *weakly* $(\alpha, H)-$*passes* $\texttt{tests}(S)$. Again by contrapositive, we will assume that I does not *weakly* $(\alpha, H)-$*pass* $\texttt{tests}(S)$ and we will conclude that $I \; \texttt{confs}_w^{(\alpha,H)} \; S$ does not hold. If I does not *weakly* $(\alpha, H)-$*pass* $\texttt{tests}(S)$ then we have two possibilities:

– Either $\texttt{pass}(I, \texttt{tests}(S))$ does not hold, or
– there exists $e = (\sigma, \bar{t}) \in \Phi$ and $T' \in \texttt{Test}(e, \texttt{tests}(S))$ such that $I \| T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ and $\gamma(\sum \zeta(s^T), \texttt{Sampling}_{(H,\Phi)}(\sigma, \bar{t})) \leq \alpha$.

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

Let us suppose now that I does not pass $\texttt{tests}(S)$ because there exists $e = (\sigma, \bar{t}) \in \Phi$ and $T' \in \texttt{Test}(e, \texttt{tests}(S))$ such that $I \parallel T \overset{\sigma}{\Longrightarrow}_{\bar{t}} s^T$ and $\gamma(\sum \zeta(s^T), \texttt{Sampling}_{(H,\Phi)}(\sigma, \bar{t})) \leq \alpha$. According to our derivation algorithm, a branch of a derived test leads to a pass state only if its associated outputs action are expected in the specification. Additionally, there exists only a tuple of random variables associated to this state, due to the specification is observable. Since $s^T \in S_P$ we have $(\sigma, \bar{t}, \zeta(s^T)) \in \texttt{InsSEvol}(S)$. Besides, by considering that $\gamma(\sum \zeta(s^T), \texttt{Sampling}_{(H,\Phi)}(\sigma, \bar{p})) < \alpha$, we conclude that $I \ \texttt{confs}_w^{(\alpha,H)} \ S$ does not hold.

Appendix 2: Hypothesis Contrasts

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

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

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

Definition 12. *Let ξ be a random variable and let J be a multiset of real numbers representing a sample. Let X^2 be the discrepancy level of J on ξ calculated as explained above by splitting the sampling space into the set of classes $C = \{[0, a_1), [a_1, a_2), \ldots, [a_{k-2}, a_{k-1}), [a_{k-1}, \infty)\}$, where k is a given constant and for all $1 \leq i \leq k-1$ we have $P(\xi \leq a_i) = \frac{i}{k}$. We define the confidence of ξ on J with classes C, denoted by $\gamma(\xi, J)$, as $\mathtt{max}\{\alpha \mid X^2 < \chi^2_\alpha(k-1)\}$.*

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

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