A statistical approach to test stochastic and probabilistic systems [⋆]

Mercedes G. Merayo[1], Iksoon Hwang[2], Manuel Núñez[1], Ana Cavalli[2]

[1] Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, 28040 Madrid, Spain,
`mgmerayo@fdi.ucm.es, mn@sip.ucm.es`

[2] Software-Networks Department, Telecom & Management SudParis
9, rue Charles Fourier, 91011, Evry Cedex, France
`Iksoon.Hwang@it-sudparis.eu, Ana.Cavalli@it-sudparis.eu`

Abstract. In this paper we introduce a formal framework to test systems where non-deterministic decisions are probabilistically quantified and temporal information is defined by using random variables. We define an appropriate extension of the classical finite state machines formalism, widely used in formal testing approaches, to define the systems that we are interested in. First, we define a conformance relation to establish, with respect to a given specification, what a *good* implementation is. In order to decide whether a system is conforming, we apply different statistic techniques to determine whether the (unknown) probabilities and random variables governing the behaviour of the implementation match the (known) ones of the specification. Next, we introduce a notion of test case. Finally, we give an alternative characterization of the previous conformance relation based on how a set of test is passed by the implementation.

1 Introduction

Formal testing techniques [20,28,5,31] 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. However, many systems require to deal with non-functional properties such as probabilities or time. On the one hand, the number of systems that incorporate nondeterminism and probabilistic behavior in order to ensure fairness and robustness in communication protocols is increasing. Some of them such as Ethernet and IEEE 802.11 have long been deployed in real networks where the exponential back-off algorithm works in a nondeterministic way. Some security protocols such as non-repudiation protocols have been proposed where the key for decrypting a message already sent is delivered with a given probability

[⋆] Research supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01), the MATES project (CCG08-UCM/TIC-4124) and by the UCM-BSCH programme to fund research groups (GR58/08 - group number 910606).

to ensure the fairness of the protocol [24]. On the other hand, the temporal behavior is critical for developing useful models of real-time systems. In fact, there are several proposals that allow to explicitly represent the probability of performing a certain task [19,10,9,7,27] as well as the time consumed by the system while performing tasks, being it either given by fix amounts of time [30,26] or defined in probabilistic/stochastic terms [15,3,21,4].

In this paper we use a suitable extension of the well known finite state machine formalism (`FSM`), *Probabilistic-Stochastic Finite State Machine (*`PSFSM`*)* introduced in [13], that allows to express in a natural way both probabilistic and temporal aspects. The *probabilities* allow to quantify the non-deterministic choices that a system may undertake. We consider a variant of the *reactive* interpretation of probabilities (see for example [19]). Intuitively, a reactive interpretation imposes a probabilistic relation among transitions labelled by the same action but choices between different actions are not quantified. In our setting we are able to express probabilistic relations between transitions outgoing from a state and having the same input action (the output may vary). The *stochastic information* represents the time consumed between the input is applied and the output is received and it will be given by *random variables.* The main idea is that time information is incremented with some kind of probabilistic information. That is, instead of having expressions such as "the message a will be received in t units of time" we will have expressions such as "the message a is 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".

There has been a lot of work to test the functional correctness of nondeterministic `FSM`s with respect to input/output sequences [11,23,29,17]. More recently models with probabilities have been studied [2,12,19,33,8,22] while some of them also include time information [18,34,25]. Nevertheless, there have been relatively few studies on testing whether the probabilities and stochastic information of a system is correctly implemented with respect to its specification. Implementation relations to assess conformance based on the observed executions of the implementation have been proposed in [22,25,14].

In this paper, we propose a methodology to test whether the probabilities and random variables of the transitions are correctly implemented. We might require that any transition of the implementation must have the same associated probability and delay, that is, an identically distributed random variable. Even though this is a very reasonable notion to define correctness, if we assume a black-box testing framework then we do not know the internal details of the implementations. So, we cannot check whether the corresponding random variables are identically distributed or the probabilities are equal to the ones established in the specification. In fact, we would need an infinite number of observations to assure it. Thus, we give a more *realistic* method based on a finite set of observations. The idea will be to check that the observed outputs and execution times in the implementation, *fit* the probabilities and random variables, respectively, established in the specification. This notion of *fitting* will be given by means of interval estimation (probabilities) and hypothesis contrasts (stochastic time).

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 conformance relations presented, in the sense that an implementation successfully passes a test suite iff it is conforming to the specification. Since the testing methodology is based on a finite set of observations, a test verdict is assigned with a given confidence level. When we test probabilities using interval estimation, different levels of quality of testing can be provided by choosing the confidence level and confidence interval length.

The rest of the paper is structured as follows. In Section 2, we define a PSFSM and notations related to testing. In Section 3 we give our conformance relations. 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. A test generation method for testing from PSFSMs is presented in Section 5. In Section 6, the basic ideas behind testing of probabilities using interval estimation and checking random variables by means of hypothesis contrast are introduced. Finally, in Section 7 we present our conclusions.

2 Preliminaries

In this section we extend the finite state machine formalism in order to deal with probabilities and stochastic time. On the one hand, probabilities attached to the transitions allow us to quantify the non-determinism of the system. On the other hand, stochastic time, represented by means of random variables, let us model the time that outputs take to be executed. We will consider that the domain of random variables is a set of numeric time values $\mathtt{Time}$. 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 $0 \in \mathtt{Time}$.

In addition, we denote by C^* the set of all finite sequences with elements in C, $\bar{c}$ denotes a sequence with length greater than 0 while ε denotes the empty sequence. Following, we introduce some basic concepts that will be used along the paper.

Definition 1. *We denote by $\mathcal{V}$ the set of random variables ($\xi, \psi, \ldots$ range over $\mathcal{V}$). Let ξ be a random variable. We define its probability distribution function as the function $F_\xi : \mathtt{Time} \longrightarrow [0,1]$ 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. Let $\xi, \xi' \in \mathcal{V}$ be random variables. We write $\xi = \xi'$ if for any $x \in \mathtt{Time}$ we have $F_\xi(x) = F_{\xi'}(x)$. We will denote by θ the random variable which probability distribution function is defined by $F(x) = 1$ for all $x \in \mathtt{Time}$.*

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 use the delimiters $\{\!|$ and $|\!\}$ to denote multisets. Given a set E, we denote by $\wp(E)$ the multisets of elements belonging to E.

A Probabilistic-Stochastic Finite State Machine is a non-deterministic finite state machine in which every transition has associated both a probability and a random variable. As we said before, the latter represents the expected distribution of times to execute the transition.

Definition 2. *A Probabilistic Stochastic Finite State Machine, in short* `PSFSM`*, is a tuple* $M = (S, s_0, Li, Lo, P_T, P_{\mathcal{V}})$ *where* S *is a finite set of states, with* $s_0 \in S$ *being the initial state,* Li *and* Lo *denote the finite input and output alphabets, respectively,* $P_T : S \times Li \times Lo \times S \to [0,1]$ *is the probability-transition function and* $P_{\mathcal{V}} : S \times Li \times Lo, S) \to \mathcal{V}$ *the time function. For all* $s \in S$ *and* $a \in Li$, $\sum_{p \in P_T(s,a,x,s')} p = 1$*. For all* $(s, a, x, s') \in S \times Li \times Lo \times S$ *if* $P_T(s, a, x, s') = p > 0$ *and* $P_{\mathcal{V}}(s, a, x, s') = \xi$ *then* (s, a, x, p, ξ, s') *is a transition of* M*.*

M *is observable if for every state* s*, input* a *and output* x *there is at most one transition leaving* s *with input* a *and output* x*.* `PSFSM` M *is completely specified if for every state* s *and input* a *there exists at least one transition outgoing from* s *and labelled with input* a*.* M *is said to be* initially connected *if every state is reachable from the initial state.*

Intuitively, a transition (s, a, x, p, ξ, s') indicates that if the machine is in state s and receives the input a then with probability p the machine emits the output x and it moves to state s' before time t with probability $F_\xi(t)$.

We do not allow that a `PSFSM` has two transitions with the same initial and final states s, s' and the same input/output a/x. Let us note that this condition does not really limit the behaviors that we can define. If we consider two different transitions, $(s, a, x, p_1, \xi_1, s')$ and $(s, a, x, p_2, \xi_2, s')$, they have the same meaning that the one provided by a unique transition (s, x, y, p, ξ, s') where $p = p_1 + p_2$ and $\xi = \frac{p_1}{p} \cdot \xi_1 + \frac{p_2}{p} \cdot \xi_2$.

Let us remark that non-deterministic choices will be resolved before the timers indicated by random variables start counting, that is, we follow a *pre-selection* policy. Thus, if we have several transitions, outgoing from a state s, associated with the same input a, and the system receives this input, then the system *at time* 0 will choose which one of them to perform according to the probabilities. So, we do not have a *race* between the different timers to decide which one is faster. In order to avoid side-effects, we will assume that all the random variables appearing in the definition of a `PSFSM` are independent. Let us note that this condition does not restrict the distributions to be used. In particular, there can be random variables identically distributed even though they are independent.

In this paper, we assume that both implementation and specification can be modeled by observable `PSFSM`s with the same input alphabet, completely specified and initially connected. If a `PSFSM` is not completely specified, it is possible to transform it to a completely specified `PSFSM` by adding a self-loop transitions for each missing input with an empty output. If the specification is not initially connected, we can consider only a sub-machine which consists of states and transitions reachable from the initial state of the system. It is also assumed that there is an upper bound on the number of states of the implementation.

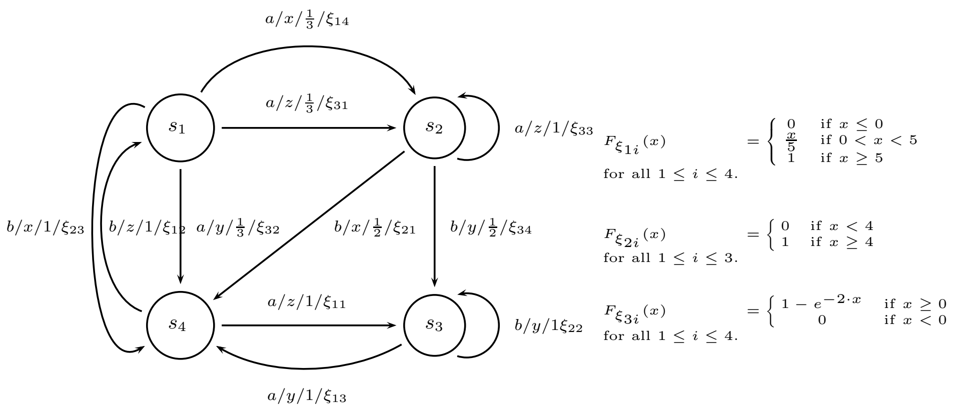
$a/x/\frac{1}{3}/\xi_{14}$
$a/z/\frac{1}{3}/\xi_{31}$
s_1
s_2
$a/z/1/\xi_{33}$
$b/x/1/\xi_{23}$
$b/z/1/\xi_{12}$
$a/y/\frac{1}{3}/\xi_{32}$
$b/x/\frac{1}{2}/\xi_{21}$
$b/y/\frac{1}{2}/\xi_{34}$
$a/z/1/\xi_{11}$
s_4
s_3
$b/y/1\xi_{22}$
$a/y/1/\xi_{13}$
$F_{\xi_{1i}}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ \frac{x}{5} & \text{if } 0 < x < 5 \\ 1 & \text{if } x \geq 5 \end{cases}$
for all $1 \leq i \leq 4$.
$F_{\xi_{2i}}(x) = \begin{cases} 0 & \text{if } x < 4 \\ 1 & \text{if } x \geq 4 \end{cases}$
for all $1 \leq i \leq 3$.
$F_{\xi_{3i}}(x) = \begin{cases} 1 - e^{-2 \cdot x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$
for all $1 \leq i \leq 4$.

Fig. 1. Example of Probabilistic Stochastic Finite State Machine.

Example 1. Let us consider the machine depicted in Figure 1. Each transition has associated a probability and a random variable. In the following we explain how these random variables are distributed. Let us consider that the random variables ξ_{1i} are *uniformly distributed* in the interval $[0, 5]$. Uniform distributions assign equal probability to all the times in the interval. The random variable ξ_{2i} follow a Dirac distribution in 4. The idea is that the corresponding delay will be equal to 4 time units. Finally, ξ_{3i} are *exponentially* distributed with parameter 2.

Let us consider the transition $(s_1, a, x, \frac{1}{3}, \xi_{14}, s_2)$. Intuitively, if the machine is in state s_1 and receives the input a then it will produce the output x with probability $\frac{1}{3}$ after a time given by ξ_{14}. 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 x has been performed (that is, we have probability 1).

The functions P_T and $P_{\mathcal{V}}$ can be extended to P_T^* and $P_{\mathcal{V}}^*$ respectively to be applied to input and output sequences.

Definition 3. *Let $M = (S, s_0, Li, Lo, P_T, P_{\mathcal{V}})$ be a* `PSFSM`. *Let $\bar{a}/\bar{x}$ be an input/output sequence and $s \in S$. We define the probability of reaching state s' from s with $\bar{a}/\bar{x}$ as:*

$$P_T^*(s, \varepsilon, x, s') = \begin{cases} 1 \text{ if } x = \varepsilon \ \wedge \ s' = s \\ \\ 0 \text{ otherwise} \end{cases}$$

$$P_T^*(s, \bar{a}a, \bar{x}x, s') = \begin{cases} P_T^*(s, \bar{a}, \bar{x}, s'') \cdot P_T(s'', a, x, s') \text{ if } \exists\, s'' \in S : & P_T(s'', a, x, s') > 0 \\ & \wedge \\ & P_T(s, \bar{a}, \bar{x}, s'') > 0 \\ 0 & \text{otherwise} \end{cases}$$

The random variable that defines the time that the system takes to reach the state s' from s performing the output sequence $\bar{x}$ if it receives the input sequence $\bar{a}$ is given by:

$$P^*_{\mathcal{V}}(s, \varepsilon, x, s') = \theta$$

$$P^*_{\mathcal{V}}(s, \bar{a}a, \bar{x}x, s') = \begin{cases} P^*_{\mathcal{V}}(s, \bar{a}, \bar{x}, s'') + P_{\mathcal{V}}(s'', a, x, s') \text{ if } \exists\, s'' \in S: & P_T(s'', a, x, s') > 0 \\ & \wedge \\ & P_T(s, \bar{a}, \bar{x}, s'') > 0 \\ \theta & \text{otherwise} \end{cases}$$

In a slight abuse of notation $P^*_T(s, \bar{a}, \bar{x})$ denotes the probability that M produces output sequence $\bar{x}$ when it receives input sequence $\bar{a}$ when in state s. In the same way, $P^*_{\mathcal{V}}(s, \bar{a}, \bar{x})$ denotes the random variable that defines the time that M takes to produce the output sequence $\bar{x}$ when it receives input sequence $\bar{a}$ when in state s.

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

Definition 4. Let $M = (S, s_0, Li, Lo, T, P_T, P_{\mathcal{V}})$ be a `PSFSM`. We say that a tuple $(s, s', (i_1/o_1, \ldots, i_r/o_r), \xi)$ is a *timed trace*, or simply *trace*, of M if there exist $(s, s_1, i_1, o_1, \xi_1)$,..., $(s_{r-1}, s', i_r, o_r, \xi_r) \in T$, such that $\xi = \sum \xi_i$.

We say that $(i_1/o_1, \ldots, i_r/o_r)$ is a *non-timed evolution*, or simply *evolution*, of M if we have that $(s_0, s', (i_1/o_1, \ldots, i_r/o_r), \xi)$ is a trace of M. We denote by $\texttt{NTEvol}(M)$ the set of non-timed evolutions of M.

We say that the pair $((i_1/o_1, \ldots, i_r/o_r), \xi)$ is a *timed evolution* of M if we have that $(s_0, s', (i_1/o_1, \ldots, i_r/o_r), \xi)$ is a trace of M. We denote by $\texttt{TEvol}(M)$ the set of timed evolutions of M. $\square$

Traces are defined as sequences of transitions. The random variable associated with the trace is computed from the corresponding to each transition belonging to the sequence. In fact, this random variables is obtained by adding the time values associated with each of the transitions conforming the trace.

3 Conformance relations for `PSFSMs`

In order to test against a specification it is necessary to say what it means for the implementation to conform to the specification. In our framework we need to consider two different levels of conformance: *functional* and *temporal*. The former only takes into account functional aspects of the system while the performance of the system, that is, the time that the system takes to perform the actions is ignored. The fact that we consider a black box framework avoid us to *see* the probabilities and the random variables assigned to the transitions in the implementation under test. In order to *estimate* the probabilities associated with each choice of the implementation we will consider a set of observations. By collecting the observations of the implementation the probabilities will be estimated by an interval with a certain level of confidence, which is called *confidence interval*, and compared with the corresponding probabilities of the specification. We define a notion of functional conformance by following the ideas underlying our methodology. Intuitively, we do not request that the probabilities of the implementation be equal to the corresponding in the specification but that this fact happens up to a certain probability.

In addition to requiring this notion of *functional* conformance, we have to ask for some conditions on delays. As we have indicated, we are not able to check whether the random variables are indeed identically distributed. Thus, we give a notion of temporal conformance based on finite sets of observations. This relation takes 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 hypothesis contrast to decide, with a certain confidence, whether the sample could be generated by the corresponding random variable.

Definition 5. *Let M be a PSFSM. We say that $(\bar{\sigma}, t)$, with $\bar{\sigma} = a_1/x_1, \ldots, a_n/x_n$, is an observed time execution of M, or simply time execution, if the observation of M shows that the time elapsed between the acceptance of the input a_1 and the observation of the output x_n is t units of time.*

Let $\Phi = \{\bar{\sigma}_1, \ldots, \bar{\sigma}_m\}$ and let $H = \{\!\{(\bar{\sigma}'_1, t_1), \ldots, (\bar{\sigma}'_n, t_n)\}\!\}$ be a multiset of timed executions. We say that $\mathit{Sampling}_{(H,\Phi)} : \Phi \longrightarrow \wp(\mathtt{Time})$ is a sampling application of H for Φ if $\mathit{Sampling}_{(H,\Phi)}(\bar{\sigma}) = \{\!\{t \mid (\bar{\sigma}, t) \in H\}\!\}$, for all $\bar{\sigma} \in \Phi$.

We say that $\mathit{SeqSampling}_{(H,\Phi)} : \Phi \longrightarrow \wp(\Phi)$ is a sequence sampling application of H for Φ if $\mathit{SeqSampling}_{(H,\Phi)}(\bar{a}/\bar{x}) = \{\!\{\bar{\sigma} \mid (\bar{\sigma}, t) \in H \wedge \bar{\sigma} = \bar{a}/\bar{x'}\}\!\}$, for all $\bar{a}/\bar{x} \in \Phi$.

Let ξ be a random variable and H be a a multiset of timed executions. We denote by $\gamma(\xi, H)$ the confidence of ξ on H. Let $0 < \alpha < 1$. We denote by $CI_\alpha(H)$ the confidence interval from H with confidence level α.

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. Regarding the definition of sampling applications, on the one hand, we assign to each sequence the total observed time corresponding to the whole execution; on the other hand, the sequence sampling application assign to each input sequence the set of output sequences observed. Let us note that $\gamma(\xi, H)$ takes values in the interval $[0, 1]$. Intuitively, bigger values of $\gamma(\xi, H)$ indicate that the observed sample H is more likely to be produced by the random variable ξ. That is, this function decides how *similar* the probability distribution function generated by H and the one corresponding to the random variable ξ are. Finally, let us note that the confidence interval length depends on the sample size and α. Larger sample size results in shorter confidence interval, which means that we can estimate the probability from the sample H. Higher values of α result in larger confidence interval and we can have higher possibility that the actual probability lies within the obtained confidence interval.

Definition 6. *Let M and M' be* PSFSMs*, H be a multiset of timed executions of M', $0 < \alpha < 1$, $\Phi = \{\bar{\sigma} \mid \exists\, t : (\bar{\sigma}, t) \in H\}$, and let us consider $\texttt{SeqSampling}_{(H,\Phi)}$. We say that M'* (α, H)-probabilistically conforms *to M, denoted by $M' \sqsubseteq_{p,(\alpha,H)} M$ if for all $\bar{\sigma} = \bar{a}/\bar{x} \in \Phi$ such that $P^*_T(s_0, \bar{a}, \bar{x}) > 0$ we have*

$$P^*_T(s_0, \bar{a}, \bar{x}) \in CI_\alpha(\texttt{SeqSampling}_{(H,\Phi)}(\bar{\sigma}))$$

Intuitively the idea is that the probabilities associated to the sequences in the M' are similar enough to the corresponding ones in M. In addition, we require conditions over the execution times.

Definition 7. *Let M and M' be* PSFSMs*, H be a multiset of timed executions of M', $0 \le \alpha \le 1$, $\Phi = \{\bar{\sigma} \mid \exists\, t : (\bar{\sigma}, t) \in H\}$, and let us consider $\texttt{Sampling}_{(H,\Phi)}$. We say that M' is* (α, H)-stochastically conformance *to M, denoted by $M' \sqsubseteq_{s,(\alpha,H)} M$ if for all $\bar{\sigma} = \bar{a}/\bar{x} \in \Phi$ we have*

$$\gamma\left(P^*_{\mathcal{V}}(s_0, \bar{a}, \bar{x}), \texttt{Sampling}_{(H,\Phi)}(\bar{\sigma})\right) > \alpha$$

The implementation M' must probabilistically conform to the specification M. Besides, for all observation, the execution time values *fit* the random variable indicated by M. This notion of *fitting* is given by the function γ that it is formally defined in the next Section.

4 Definition and application of tests

We consider that tests represent sequences of inputs applied to an implementation. 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 L_I and L_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, we have to check if the implementation behaves according to probabilities established in the specification. We have also to detect whether wrong timed behaviors appear. Thus, tests have to include capabilities to deal with probabilities and time. On the one hand, tests will include *probabilities*. In our proposal, we will *estimate* the probabilities by applying a test several times and we will use *statistical results* to establish the number of times we need to apply the test to obtaining a required confidence level. On the other hand, we will include *random variables*. 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.

Definition 8. A *test case* is a tuple $T = (S, L_I, L_O, \lambda, s_0, S_I, S_O, S_F, S_P, \zeta, \eta)$ where S is the set of states, I and O, with $L_I \cap L_O = \emptyset$ are the sets of input and output actions, respectively, $\lambda \subseteq S \times L_I \cup L_O \times S$ 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 $a \in L_I$ 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 $i \in L_I, s' \in S$ 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 $a \in L_I \cup L_O$ and $s' \in S$ such that $(s, a, s') \in \lambda$.

Finally, we have two timed functions. $\zeta : S_P \longrightarrow \mathcal{V}$ is a function associating random variables, to compare with the time that took to perform the outputs, with passing states. $\eta : S_P \longrightarrow (0, 1]$ is a function associating probabilities with passing 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 $\mathcal{T}_{st} = \{T_1, \dots, T_n\}$ is a *test suite*.

Let $\bar{\sigma} = i_1/o_1, \dots, i_r/o_r$. We write $T \overset{\bar{\sigma}}{\Longrightarrow} s^T$ 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 us remark that $T \overset{\bar{\sigma}}{\Longrightarrow} s^T$ implies 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.

Definition 9. Let I be `PSFSM` and $T = (S_t, L_I, L_O, \lambda_T, s_0, S_I, S_O, S_F, S_P, \zeta, \eta)$ be a valid test, $\bar{\sigma} = i_1/o_1, \dots, i_r/o_r$, s^T be a state of T, and $\bar{t} = (t_1, \dots, t_r)$.We write $I \parallel T \overset{\bar{\sigma}}{\Longrightarrow} s^T$ if $T \overset{\bar{\sigma}}{\Longrightarrow} s^T$ and $\bar{\sigma} \in \mathtt{NTEvol}(I)$.

We write $I \| T \overset{\bar{\sigma}}{\Longrightarrow}_{\bar{t}} s^T$ if $I \| T \overset{\bar{\sigma}}{\Longrightarrow} s^T$ and $(\bar{\sigma}, \bar{t})$ is a observed timed execution of I. In this case we say that $(\bar{\sigma}, \bar{t})$ 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 $\bar{\sigma} \in \mathtt{NTEvol}(I)$, $s^T \in S$ such that $I \parallel T \overset{\bar{\sigma}}{\Longrightarrow} s^T$ and $s^T \in S_F$. □

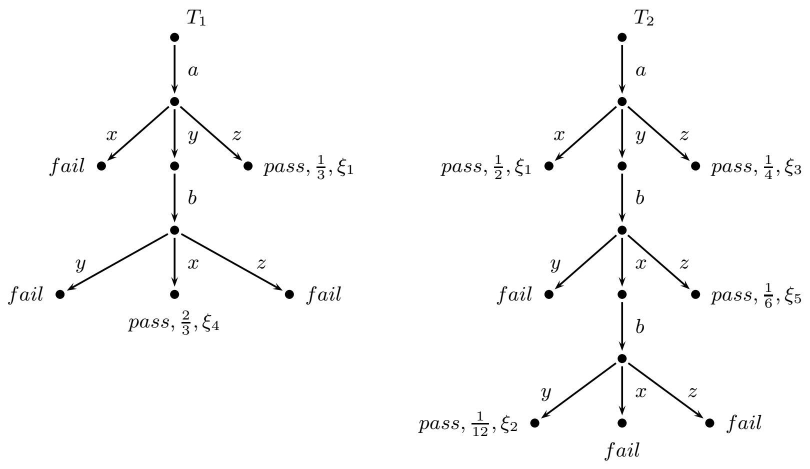
T_1
a
x
y
z
$fail$
$pass, \frac{1}{3}, \xi_1$
b
y
x
z
$fail$
$pass, \frac{2}{3}, \xi_4$
$fail$
T_2
a
x
y
z
$pass, \frac{1}{2}, \xi_1$
$pass, \frac{1}{4}, \xi_3$
b
y
x
z
$fail$
$pass, \frac{1}{6}, \xi_5$
b
y
x
z
$pass, \frac{1}{12}, \xi_2$
$fail$
$fail$

Fig. 2. Examples of Test Cases: $I = \{a, b\}$ and $O = \{x, y, z\}$.

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 probabilistic and time conditions. We apply these conditions to the set of *observed timed executions*, not to evolutions of the implementations. In fact, we need a set of test executions associated to each evolution to evaluate if they match these conditions. 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 PSFSM and $\mathcal{T}_{st} = \{T_1, \ldots, T_n\}$ be a test suite. Let $H_1, \ldots, H_n$ be test execution samples of I and $T_1, \ldots, T_n$, respectively. Let $H = \bigcup_{i=1}^{n} H_i$, $\Phi = \{\bar{\sigma} \mid \exists\, \bar{t} : (\bar{\sigma}, \bar{t}) \in H\}$, and $\bar{\sigma} \in \Phi$. We define the set $\texttt{Test}(\bar{\sigma}, \mathcal{T}_{st}) = \{T \mid T \in \mathcal{T}_{st} \wedge I \parallel T \overset{\bar{\sigma}}{\Longrightarrow} s^T\}$.*

Let us consider $0 < \alpha < 1$, $\texttt{Sampling}_{(H,\Phi)}$ and $\texttt{SeqSampling}_{(H,\Phi)}$. We say that the implementation I probabilistically $(\alpha, H)-$passes the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(I, \mathcal{T}_{st})$ and for all $\bar{\sigma} = \bar{a}/\bar{x} \in \Phi$ we have that for all $T \in \texttt{Test}(\bar{\sigma}, \mathcal{T}_{st})$ it holds $\eta(s^T) \in CI_\alpha(\texttt{SeqSampling}_{(H,\Phi)}(\bar{\sigma}))$.

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

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 probabilistic and 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 $\bar{\sigma}$, the frequency (that is, $\texttt{SeqSampling}_{(H,\Phi)}(\bar{\sigma})$) and the observed time values (that is, $\texttt{Sampling}_{(H,\Phi)}(\bar{\sigma})$) *match* the probabilities and the definition of the random variables appearing in the successful state of the tests corresponding to the execution of that evolution (that is, $\eta(s^T)$ and $\zeta(s^T)$).

5 Derivation of Test Suites

In this section we present an algorithm to derive tests from specifications. We will derive test suites that are sound and complete with respect to the implementation relations introduced in Section 3. The basic idea underlying test derivation consists in traversing the specification in order to get all the possible evolutions in an appropriate way. First, we introduce some additional notation.

Definition 11. Let $M = (S, s_0, Li, Lo, P_T, P_{\mathcal{V}})$ be a PSFSM. We define the function $\texttt{out} : S \times Li \longrightarrow \wp(Lo)$ such that for all $s \in S$ and $i \in I$ it returns the set of outputs

$$\texttt{out}(s,a) = \{x \mid \exists\, s', p, \xi :\ P_T(s,a,x,s') = p > 0\ \wedge\ P_{\mathcal{V}}(s,a,x,s') = \xi\}$$

We define the function $\texttt{after} : S \times Li \times Lo \times \mathcal{V} \times [0,1] \longrightarrow ((S \times \mathcal{V} \times [0,1]) \cup \{\texttt{error}\})$ such that for all $s \in S, a \in Li, x \in Lo$, $p \in [0,1]$ and $\xi \in \mathcal{V}$ we have

$$\texttt{after}(s,a,x,\xi,p) = \begin{cases} (s', \xi + \xi', p \cdot p') & \text{if}\ \ \exists\, s', p', \xi' : P_T(s,a,x,s') = p' > 0\ \wedge \\ & \qquad P_{\mathcal{V}}(s,a,x,s') = \xi' \\ \texttt{error} & \qquad \text{otherwise} \end{cases}$$

□

The function $\texttt{out}(s,a)$ computes the set of output actions associated with those transitions that can be executed from s after receiving the input a. The function $\texttt{after}(s,a,x,\xi,p)$ computes the *situation* that is reached from a state s after receiving the input a, producing the output x, when the duration of the previous testing process is given by ξ with probability p. Let us also remark that due to the assumption that PSFSMs are observable we have that $\texttt{after}(s,a,x,\xi,p)$ is uniquely determined. Finally, we will apply this function only when the side condition holds, that is, we will never receive error as result of applying after.

The algorithm to derive tests from a specification is given in Figure 3. By considering the possible available choices we get a test suite extracted from the specification M. We denote this test suite by $\texttt{tests}(M)$.

Next we explain how our algorithm works. A set of *pending situations* S_{aux} keeps those tuples denoting the possible states, random variables and probabilities that could appear in a state of the test whose outgoing transitions have not

```
Input: A specification $M = (S, s_{in}, Li, Lo, P_T, P_{\mathcal{V}})$.
Output: A test case $T = (S, Li, Lo, \lambda, s_0, S_I, S_O, S_F, S_P, \zeta, \eta)$.

Initialization: $S' := \{s_0\};\ Tran', S_I, S_O, S_F, S_P := \emptyset;\ S_{aux} := \{(s_{in}, \theta, 0, s_0)\}$.
Inductive Cases: Choose one of the following two options until $S_{aux} = \emptyset$.

1. If $(s^M, \xi, p, s^T) \in S_{aux}$ then perform the following steps:
   (a) $S_{aux} := S_{aux} - \{(s^M, \xi, p, s^T)\};\ S_P := S_P \cup \{s^T\};\ \zeta(s^T) := \xi;\ \eta(s^T) := p$.
2. If $S_{aux} = \{(s^M, \xi, p, s^T)\}$ and $\exists\, a \in Li : \mathtt{out}(s^M, a) \neq \emptyset$ then perform the following
   steps:
   (a) $S_{aux} := \emptyset$; Choose $a$ such that $\mathtt{out}(s^M, a) \neq \emptyset$.
   (b) Consider a fresh state $s' \notin S'$ and let $S' := S' \cup \{s'\}$.
   (c) $S_I := S_I \cup \{s^T\};\ S_O := S_O \cup \{s'\};\ Tran' := Tran' \cup \{(s^T, a, s')\}$.
       {Add an input transition labelled by $a$ and consider all outputs}
   (d) For all $x \notin \mathtt{out}(s^M, a)$ do {These outputs lead to a fail state}
         – Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
         – $S_F := S_F \cup \{s''\};\ Tran' := Tran' \cup \{(s', x, s'')\}$.
   (e) For all $x \in \mathtt{out}(s^M, a)$ do
       {These outputs are expected. At most one of them will lead to an
       input state where the test continues; the rest will lead to pass
       states}
         – Consider a fresh state $s'' \notin S'$ and let $S' := S' \cup \{s''\}$.
         – $Tran' := Tran' \cup \{(s', x, s'')\}$.
         – $(s_1^M, \xi', p') := \mathtt{after}(s^M, a, x, \xi, p);\ S_{aux} := S_{aux} \cup \{(s_1^M, \xi', p', s'')\}$.
```

Fig. 3. Derivation of tests from an observable specification.

been completed yet. More precisely, a tuple $(s^M, \xi, p, s^T) \in S_{aux}$ indicates that we did not complete the state s^T of the test, the current state in the traversal of the specification is s^M, and the accounting for the elapsed duration in the specification from the initial state is given by ξ. In addition, p reflects the probability associated to the transitions traversed in the specification for reaching the current state.

Following with the explanation of the algorithm, the set S_{aux} initially contains a tuple with the initial states (of both specification and test) and the initial situation of the process, that is, duration θ and probability 0. For each tuple belonging to S_{aux} we may choose one possibility. It is important to remark that the second step can be applied only when the set S_{aux} becomes singleton. So, our derived tests correspond to valid tests as introduced in Definition 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. Then, the whole sets of outputs is considered. If the output is not expected by the specification (step 2.(d) 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.(e) of the algorithm) we create a transition with the corresponding output and add the appropriate tuple to the set S_{aux}.

Let us note that finite tests are constructed simply by considering a step where the second inductive case is not applied. The algorithm provide us with a test suite extracted from the specification S that we denote by $\texttt{tests}(S)$.

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 *fault coverage criterion* to generate a finite test suite. Obviously, one can impose restrictions such as "generate n tests" or "generate all the tests with m inputs" and *completeness* will be obtained up to that coverage criterion. Since we do not assume, by default, any criteria, all we can do is to say that this is the, in general, test suite that allows to prove completeness, that is, we obtain full fault coverage but taking into account that the derived test suite will be, in general, infinite.

Next, we present the results that relate implementation relations and application of test suites derived from a specification. The result holds because the temporal and probabilistic conditions required to conform to the specification and to pass the test suite are in fact the same. Due to space limitations, we cannot include in this paper the proof of the theorem. In spite of the differences, the proof is an adaptation of that in [25].

Theorem 1. (Soundness and Completeness) Let I and S be `PSFSM`s, H be a multiset of timed executions of I, $0 < \alpha < 1$. We have that:

- $I \sqsubseteq_{p,(\alpha,H)} S$ iff I *probabilistically* $(\alpha, H)-$*passes* $\texttt{tests}(S)$.
- $I \sqsubseteq_{s,(\alpha,H)} S$ iff I *stochastically* $(\alpha, H)-$*passes* $\texttt{tests}(S)$.

□

The derived test suite is *sound* and *complete*, up to a given confidence level α and for a sample H, with respect to the conformance relations $\sqsubseteq_{s,(\alpha,H)}$ and $\sqsubseteq_{p,(\alpha,H)}$. Specifically, for a given specification S, the test suite $\texttt{tests}(S)$ can be used to distinguish those (and only those) implementations that conform with respect to $\sqsubseteq_{s,(\alpha,H)}$ and $\sqsubseteq_{p,(\alpha,H)}$. However, we cannot say that the test suite is complete since both the notion of passing tests and the considered implementation relations have a probabilistic component. So, we can talk of *completeness* up to a certain confidence level.

6 Estimation of probabilities and random variables

When testing from a non-deterministic `FSM`s, it is necessary to make an assumption: *implementations have a fairness property such that if an input sequence is applied to the implementation a finite number of times, all possible execution* *paths in the implementation that can be followed using the input sequence are traversed.* This is the so-called *complete-testing assumption.* Under this assumption, we need to apply the test sequences a minimum number of times to the implementation in order to observe all output sequences that can be produced. In addition, in our approach, we need to estimate the probabilities associated to each input/output sequence in order to determine if they fulfill the requirements specified. Moreover, we need to check if the random variables are correctly implemented. In order to do it we collect a set of timed executions corresponding to the observations of the several applications of the tests belonging to a test suite to the implementation. Nevertheless, this technique does not allow us to determine the probabilities and the random variables of the implementation. We only can *estimate* the probabilities and decide with a certain confidence, whether the sample could be generated by the corresponding random variable in the specification. We use *statistical results* to establish the number of times we need to apply the test to obtaining a required confidence level.

6.1 Checking correctness of probabilities

In general, test sequence repetition numbers are neither large enough to satisfy the complete-testing assumption nor large enough to estimate exact probabilities. In such a case, the testing process is a hypothesis test [32]. When we check the probability of a given input/output sequence the following two hypotheses are considered in this paper.

H_0 : *the probability of the implementation is correct*
H_1 : *the probability of the implementation is not correct.*

Let us denote by P_{EP} and P_{NEF} test-pass probability of equivalent machines and test-fail probability of faulty machines respectively. $(1-P_{EP})$ corresponds to type I error of hypothesis H_0 which is the probability that the hypothesis H_0 is rejected when it is true. $(1-P_{NEF})$ corresponds to type II error of hypothesis H_0 which is the probability that the hypothesis H_0 is accepted when it is false. When we test probabilities using interval estimation, the numbers of test application will be determined such that P_{EP} and P_{NEF} are not not less than a given value where the types of faulty machines can be described further in hypothesis H_1.

If there are two executable transitions t_1 and t_2 from a state s when an input is applied where the transition probabilities are p and $1-p$ respectively, a random variable X defined as follows is a Bernoulli random variable:

$$\begin{cases} X = 0 \text{ if transition } t_1 \text{ is selected for execution from state } s; \\ X = 1 \text{ if transition } t_2 \text{ is selected for execution from state } s. \end{cases}$$

Selecting a transition between t_1 and t_2 for execution can be considered as an *experiment* (or a *trial*). If Y represents the number of times t_1 was selected after n independent experiments, Y is said to be a *binomial* random variable with parameters (n, p). The observed data after n independent experiments is called a *sample* where n is the *sample size.* p can be estimated by an interval with a certain degree of confidence, which is called the *confidence interval*, after n independent experiments. The Agresti-Coull interval [1] which is one of the recommended intervals by Brown *et al.* [6] for p with $100(1-\alpha)\%$ confidence is given by

$$\tilde{p} - \kappa\sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}} \leq p \leq \tilde{p} + \kappa\sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}}$$

where $\tilde{Y} = Y + \kappa^2/2$, $\tilde{n} = n + \kappa^2$, $\tilde{p} = \tilde{Y}/\tilde{n}$, and κ is such that $P\{|Z| \leq \kappa\} = 1-\alpha$ where Z is a standard normal variable. For the case when α=0.05, the value 2 is used instead of 1.96 for κ in the Agresti-Coull interval. If $n \geq 40$, the Agresti-Coull interval provides good coverage even for p very close to 0 or 1 [6].

Suppose that an implementation PSFSM M_I has a transition with probability p' while the probability in the specification PSFSM M is p. The following criteria can be used for testing:

$$\begin{cases} \textit{Pass} \text{ if } p \text{ is included in the obtained confidence interval for } p'; \\ \textit{Fail} \text{ otherwise.} \end{cases}$$

Let d denote half of the obtained confidence interval length. According to the value of p' we will have the following results. If $p' = p$, $100(1-\alpha)\%$ of the time the implementation will have a pass verdict. When we test probabilities using interval estimation, therefore, we can ensure that P_{EP} is never less than a given value, $(1-\alpha)$. If $|p'-p| > 2d$, $100(1-\alpha/2)\%$ of the time, the implementation will have a fail verdict and P_{NEF} is ensured to be not less than a given value, $(1-\alpha/2)$. If $0 < |p'-p| \leq 2d$, we cannot provide any meaningful upper or lower bound of P_{NEF} as the range is too wide, from α to $(1-\alpha/2)$ according the difference between p and p'.

We now explain how to determine test sequence repetition numbers for testing probabilities. Test sequence repetition numbers will be determined so that P_{EP} and P_{NEF} are ensured to be not less than $(1-\alpha)$ and $(1-\alpha/2)$ respectively where faulty implementations are such that the difference of the probability is more than $2d$. For correct implementations, ideally, $100(1-\alpha)\%$ of the time, p will be contained in any size of confidence interval. Therefore, P_{EP} is independent of d and we can always ensure P_{EP} as far as we have a reliable confidence interval. In order to have $P_{NEF} \geq 1-\alpha/2$, the test sequence repetition number n should satisfy the following condition.

$$n > \left(\frac{\kappa}{d}\right)^2 \hat{p}(1-\hat{p}) - \kappa^2$$

where

$$\hat{p} = \begin{cases} p + d \text{ if } p \leq 0.5 \\ p - d \text{ otherwise} \end{cases}$$

If there are j probabilities to test in a test sequence ts and there are k test sequences in a test suite which have at least two probabilities to test, κ should satisfy the following condition when we test the probabilities of the test sequence.

$$P\{|Z| < \kappa\} = \begin{cases} (1-\alpha)^{1/k} & \text{if } j = 2 \\ (1-\alpha)^{1/(j+k-1)} & \text{if } j > 2 \end{cases}$$

For further details about the determination of test sequence repetition numbers for testing probabilities can be found at [16].

6.2 Checking correctness of random variables

Goodness-of-fit tests indicate whether or not it is sensible to assume that a random sample comes from a specific distribution. Hypothesis Test model for Goodness-of-fit tests are a form of hypothesis testing where the null and alternative hypotheses are

- H_0: Sample data come from the stated distribution.
- H_A: Sample data do not come from the stated distribution.

The underlying idea is that a sample will be *rejected* if the probability of observing that sample from a given random variable is low. In practice, we will check whether the probability to observe a *discrepancy* lower than or equal to the one we have observed is low enough.

Three goodness-of-fit tests are the most frequently used. Chi-square test can be applied for both continuous and discrete distributions. Kolmogorov-Smirnov and Anderson-Darling test only can be used for continuous distributions. Due to the fact that our models may present random variables associated to discrete and continuous distributions functions, we will present a methodology based on Chi-square test to measure the confidence degree that a random variable has on a sample.

The mechanism is the following. Once we have collected a sample of size n we perform the following steps:

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

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

Definition 12. *Let ξ be a random variable and J be a multiset of real numbers representing a sample. Let X^2 be the discrepancy level of J on ξ calculated as explained above by splitting the sampling space into k classes*

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

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

The previous definition indicates that in order to perform a contrast hypothesis, we split the collected values in several intervals having the same expected probability. We compute the value for X^2 as previously described and check this figure with the tabulated tables corresponding to χ^2 with $k-1$ freedom degrees (see, for example, `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, the number of classes and how many data contain each class will affect the power of the test, that is, how sensitive it is to detecting departures from the null hypothesis. Power will not only be affected by the number of classes and how they are defined, but by the sample size and shape of the null and underlying distributions. Some useful rules can be given in order to determine it. When data are discrete, tabulation can be used to categorize the data. Continuous data present a more difficult challenge. One defines classes by segmenting the range of possible values into non-overlapping intervals. Elements can then be defined by the endpoints of the intervals. In general, power is maximized by choosing endpoints such that each element is equiprobable, that is, the probabilities associated with an observation falling into a given class are divided as evenly as possible across the intervals. A good starting point for choosing the number of classes is to use the value $2 \cdot n^2/5$, each of them containing at least 5 elements.

Example 2. Let us consider a device that produces real numbers belonging to the interval $[0, 1]$. We would like to test whether the device produces these numbers randomly, that is, it does not have a number or sets of numbers that have a higher probability of being produced than others. Thus, we obtain a sample from the machine and we apply the contrast hypothesis to determine whether the machine follows a uniform distribution in the interval $[0, 1]$. First, we have to decide how many classes we will use. Let us suppose that we take $k = 10$ classes. Thus, for all $1 \leq i \leq 9$ we have $a_i = 0.i$ and $P(\xi \leq a_i) = \frac{i}{10}$. So, $C = \{[0, 0.1), [0.1, 0.2) \ldots [0.8, 0.9), [0.9, \infty)\}$.

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

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

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

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

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

7 Conclusions

In this paper we have presented a notion of finite state machine to specify, in an easy way, both probabilities that quantified the non-deterministic choices and the passing of time due to the performance of actions. In addition, we have presented two implementation relations based on the notion of conformance. First, the implementation must conform to the specification regarding functional aspects. It requires to take into account the probabilities associated to the transitions in the specification. Second, we require that the implementation complies with the temporal requirements specified by means of random variables. In order to check that the implementation fulfills the probabilities and random variables established in the specification, we apply statistical results. Additionally, 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 to check the correctness of an implementation, applying a derived test suite.

References

1. A. Agresti and B.A. Coull. Approximate is better than exact for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998.
2. R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *27th ACM Symp. on Theory of Computing, STOC'95*, pages 363–372. ACM Press, 1995.
3. 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.
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. L. D. Brown, T. T. Cai, and A. Dasgupta. Interval estimation for a binomial proportion. *Statistical Science*, 16:101–133, 2001.
7. 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.
8. L. Cheung, M. Stoelinga, and F. Vaandrager. A testing scenario for probabilistic processes. *Journal of the ACM*, 54(6):Article 29, 2007.
9. R. Cleaveland, Z. Dayar, S.A. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93–148, 1999.
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. R.M. Hierons. Testing from a non-deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.
12. R.M. Hierons and M.G. Merayo. Mutation testing from probabilistic finite state machines. In *3rd Workshop on Mutation Analysis, Mutation'07*, pages 141–150. IEEE Computer Society Press, 2007.
13. R.M. Hierons and M.G. Merayo. Mutation testing from probabilistic and stochastic finite state machines. *Journal of Systems and Software (in press)*, 2009.
14. R.M. Hierons, M.G. Merayo, and M. Núñez. Testing from a stochastic timed system with a fault model. *Journal of Logic and Algebraic Programming*, 78(2):98–115, 2009.
15. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
16. I. Hwang and A. Cavalli. Testing from a probabilistic FSM using interval estimation. Technical Report 09004LOR, TELECOM & Management SudParis, 2009.
17. I. Hwang, T. Kim, S. Hong, and J. Lee. Test selection for a nondeterministic FSM. *Computer Communications*, 24(12):1213–1223, 2001.
18. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.
19. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
20. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

21. N. López and M. Núñez. A testing theory for generally distributed stochastic processes. In *12th Int. Conf. on Concurrency Theory, CONCUR'01, LNCS 2154*, pages 321–335. Springer, 2001.
22. 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.
23. G.L. Luo, G. von Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–161, 1994.
24. O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In *2nd Conf. on Security in Communication Network*, 1999.
25. M.G. Merayo, M. Núñez, and I. Rodríguez. Formal testing from timed finite state machines. *Computer Networks*, 52(2):432–460, 2008.
26. 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.
27. M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.
28. 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.
29. A. Petrenko, N. Yevtushenko, and G. von Bochmann. Testing deterministic implementations from their nondeterministic FSM specifications. In *9th IFIP Workshop on Testing of Communicating Systems, IWTCS'96*, pages 125–140. Chapman & Hall, 1996.
30. G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.
31. I. Rodríguez, M.G. Merayo, and M. Núñez. $\mathcal{HOTL}$: Hypotheses and observations testing logic. *Journal of Logic and Algebraic Programming*, 74(2):57–93, 2008.
32. S.M. Ross. *Introduction to Probability and Statistics for Engineers and Scientists.* John Wiley & Sons, 1987.
33. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
34. M.Ü. Uyar, S.S. Batth, Y. Wang, and M.A. Fecko. Algorithms for modeling a class of single timing faults in communication protocols. *IEEE Transactions on Computers*, 57(2):274–288, 2008.