Testing of Symbolic-Probabilistic Systems*

Natalia López, Manuel Núñez, and Ismael Rodríguez

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

Abstract. We consider the testing of systems where probabilistic information is not given by means of fixed values but as sets of probabilities. We will use an extension of finite state machine where choices among transitions having the same input actions are probabilistically resolved. We will introduce our notion of test and we will define how tests are applied to the implementation under test (IUT). We will also present implementation relations to assess the conformance, *up to* a level of confidence, of an implementation to a specification. In order to define these relations we will take finite *samples* of executions of the implementation and compare them with the probabilistic constraints imposed by the specification. Finally, we will give an algorithm for deriving sound and complete test suites.

1 Introduction

During the last years we have seen an evolution in the kind of systems that formal methods are dealing with. In the beginning the main focus was on functional properties. The next step was to consider quantitative information such as the *time* underlying the performance of the system or the *probabilities* resolving the non-deterministic choices to be taken. Thus, plenty of proposals considering time and/or probabilistic aspects have appeared (e.g. [15,12,19,6,7,1,4,13,9]).

In this paper we consider the testing of a novel notion of probabilistic systems. One of the main criticisms about formal models including probabilistic information is the inability, in general, to accurately determine the actual values of the involved probabilities. In fact, using a process algebraic notation, the usual inclusion of probabilistic information is given by processes such as $P = a +_{\frac{1}{3}} b$. Intuitively, the process P indicates that if the choice between a and b must be resolved then a has probability $\frac{1}{3}$ of being chosen while b has probability $\frac{2}{3}$. However, there are situations where it is rather difficult to be so precise when specifying a probability. A very good example is the specification of *faulty channels* (e.g. the classical ABP [2]). Usually, these protocols contain information such as "the probability of losing the message is equal to 0.05." However, two questions may be immediately raised. First, why would one like to specify the exact probability of

* Work supported by the Spanish MCyT project *MASTER* (TIC2003-07848-C02-01) and the Junta de Castilla-La Mancha project *DISMEF* (PAC-03-001).

losing a message? Second, how can the specifier be sure that this is in fact the probability? In other words, to know the exact value seems as a very strong requirement. It would be more appropriate to say "the probability of losing the message is smaller that 0.05." Such a statement can be interpreted as: "we know that the probability is low but we do not know the exact value." Moreover, this kind of probabilistic information, that we call *symbolic probabilities*, allows us to *refine* the model. For example, let us suppose that after experimentation with the system we have detected that some messages are lost (so that we can discard that the probability of losing a message is equal to zero) but *not many* messages were lost. By using the appropriate statistics machinery (mainly hypothesis contrasts and Tchebyshev inequality) we may infer information such as "with probability 0.95 the *real* probability of losing the message belongs to the interval $[0.01, 0.03]$."

Let us remark that probabilistic information will be not be the same for specifications and implementations. In the former case we might allow the specifier to use symbolic probabilities. In contrast, implementations will have fixed probabilities governing their behavior. For example, we may specify a *not-very-unfair coin* as a coin such that the probability of obtaining tails belongs to the interval $[0.4, 0.6]$ (and the same for faces). Given a *real* coin (i.e. an implementation) the probability p_t of obtaining tails (resp. p_f for faces) will be a fixed number (possibly unknown, but fixed). If $p_t, p_c \in [0.4, 0.6]$ then we will consider that the implementation conforms to the specification.

Our testing methodology follows a black-box testing approach (see e.g. [11,3]). That is, if we apply an input to an IUT then we will observe an output and we may continue the testing procedure according to this result. However, we will not be able to *see* the probabilities that the IUT has assigned to each of the choices. Thus, even though implementations will behave according to fixed probabilities we will not be able to *read* their values. In order to compute the probabilities associated with each choice of the implementation we will apply the same test several times and analyze the obtained responses. The set of tests used to check the suitability of an implementation will be constructed from the given specification. By collecting the observations and comparing them with the symbolic probabilities of the specification, we will be able to assess the validity of the IUT. This comparison will be performed by using *hypothesis contrasts*. Hypothesis contrasts allow to (probabilistically) decide whether an observed sample follows the pattern given by a random variable. For example, even if we do not know the exact probabilities governing a *coin under test*, if we toss the coin 1000 times and we get 502 faces and 498 tails then we can infer, with a big probability, that the coin conforms with the specification of *not-very-unfair coin*.

In order to specify probabilistic systems dealing with symbolic probabilities we will consider the probabilistic extension of *finite state machines* recently introduced in [10]. Intuitively, transitions in finite state machines indicate that if the machine is in a state s and receives an input i then it will produce an 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 probabilistic extension of finite state machines, a transition such as $s \xrightarrow{i/o}_p s'$ indicates that the probability with which the previ-

ous sequence of events happens is equal to p. By using symbolic probabilities we go one step further. A transition such as $s \xrightarrow{i/o}_{\bar{p}} s'$ indicates that the probability with which the corresponding transition is performed belongs to the range given by $\bar{p}$. For instance, $\bar{p}$ may be the interval $[\frac{1}{4}, \frac{3}{4}]$ while the *real* probability is in fact 0.53.

An important issue when dealing with probabilities consists in fixing how different actions/transitions are related according to the probabilistic information. In this paper we consider a variant of the *reactive* interpretation of probabilities (see for example [8]) since it is the most suitable for our framework. Intuitively, a reactive interpretation imposes a probabilistic relation among transitions labeled by the same action, but without quantifying choices between different actions. Our probabilistic finite state machines are able to express probabilistic relations between transitions outgoing from a given state and having the same input action (while the output action may vary). Thus, we are assuming a kind of *reactive* interpretation of probabilities. In the following example we illustrate this notion (a formal definition will be given in the next section). Let us consider that the unique transitions from a state s are

$$t_1 = s \xrightarrow{i_1/o_1}_{\bar{p}_1} s_1 \qquad t_2 = s \xrightarrow{i_1/o_2}_{\bar{p}_2} s_2 \qquad t_3 = s \xrightarrow{i_1/o_3}_{\bar{p}_3} s_2$$
$$t_4 = s \xrightarrow{i_2/o_1}_{\bar{p}_4} s_3 \qquad t_5 = s \xrightarrow{i_2/o_3}_{\bar{p}_5} s_1$$

If the environment (in our case, if the test) offers the input action i_1 then the choice between t_1, t_2, and t_3 will be resolved according to some probabilities fulfilling the conditions $\bar{p}_1$, $\bar{p}_2$, and $\bar{p}_3$. All we know about these values is that they fulfill the imposed restrictions, that they are non-negative, and that the sum of them equals 1. Something similar happens for the transitions t_4 and t_5. However, there does not exist any probabilistic relation between transitions labeled with different input actions (e.g. t_1 and t_4).

In addition to our testing methodology, we will also introduce new implementation relations. In [10] we presented two implementation relations where we required that the probabilities governing the implementation belong to the corresponding intervals of the specification. Unfortunately, these notions are useful only from a theoretical point of view. This is so because it is not possible to check, by using a finite number of observations, the correctness of the probabilistic behavior of an implementation with respect to a specification. The new implementation relations are defined following the ideas underlying our testing methodology. Intuitively, in this paper we do not request that the probabilities of the implementation belong to the corresponding intervals of the specification but that this happens *up to a certain probability*.

The rest of the paper is organized as follows. In Section 2 we review our probabilistic finite state machines model. In Section 3 we present the notion of test and define how they are applied to implementations. In Section 4 we introduce two implementation relations based on samples. The underlying idea is that a hypothesis contrast is used to assess whether the observed behavior corresponds, *up to* a certain confidence, to the probabilistic behavior defined in the specification. In Section 5 we present an algorithm to derive sound and complete test suites with

respect to one of the relations presented in the previous section. In fact, we cannot properly use the term *completeness* but *completeness up to a certain confidence level*. In Section 6 we present our conclusions and some lines for future work. Finally, in the appendix of this paper we give some basic statistical concepts that are (abstractly) used along the paper (this appendix is extracted from [10]).

2 Probabilistic Finite State Machines

In this section we introduce our notion of probabilistic finite state machines. As we have previously mentioned, probabilistic information will not be given by fixed values of probabilities but by introducing certain constraints on the considered probabilities. By taking into account the inherent nature of probabilies, we will consider that these constraints are given by intervals contained in $(0, 1] \subseteq \mathbb{R}$.

Definition 1. We define the set of *symbolic probabilities*, denoted by `simbP`, as the following set of intervals

$$\texttt{simbP} = \left\{ \$p_1, p_2\& \;\middle|\; \begin{array}{l} p_1, p_2 \in [0,1] \;\wedge\; p_1 \leq p_2 \;\wedge\; \$ \in \{\,(,[\,\} \;\wedge\; \& \in \{\,),]\,\} \wedge \\ 0 \notin \$p_1, p_2\& \;\wedge\; \$p_1, p_2\& \neq \emptyset \end{array} \right\}$$

If we have a symbolic probability as $[p, p]$, with $0 < p \leq 1$, we simply write p.

Let $\overline{p}_1, \ldots, \overline{p}_n \in \texttt{simbP}$ be symbolic probabilities such that for any $1 \leq i \leq n$ we have $\bar{p}_i = \$_i p_i, q_i \&_i$, with $\$_i \in \{\,(,[\,\}$ and $\&_i \in \{\,),]\,\}$. We define the *product* of $\overline{p}_1, \ldots, \overline{p}_n$, denoted by $\prod \overline{p}_i$, (respectively the *addition* of $\overline{p}_1, \ldots, \overline{p}_n$, denoted by $\sum \overline{p}_i$) as the symbolic probability $\$ \prod p_i, \prod q_i \&$ (respectively $\$ \sum p_i, \sum q_i \&$). The limits of the interval are defined in both cases as:

$$\$ = \begin{cases} (& \text{if } \exists 1 \leq i \leq n : \; \$_i = (\\ [& \text{otherwise} \end{cases} \qquad \& = \begin{cases}) & \text{if } \exists 1 \leq i \leq n : \; \&_i =) \\ ] & \text{otherwise} \end{cases}$$

□

The previous definition expresses that a symbolic probability $\overline{p}$ is any non-empty (open or closed) interval contained in $(0, 1]$. In particular, we will not allow transitions with probability 0 because this value would complicate (even more) our model since we would have to deal with priorities.[1] We have also defined how to *multiply* and *add* symbolic probabilities. The maximal (resp. minimal) bound of the resulting interval is obtained by operating over the maximal (resp. minimal) bounds of the considered intervals. Next, we introduce our notion of probabilistic finite state machine. In the following definition we assume that $|\, X \,|$ denotes the cardinality of the set X.

Definition 2. A *Probabilistic Finite State Machine*, in short `PFSM`, is a tuple $M = (S, I, O, \delta, s_0)$ where S is the set of states, I and O denote the sets of input and output actions, respectively, $\delta \subseteq S \times I \times O \times \texttt{simbP} \times S$ is the set of

[1] The interested reader can check [5] where different approaches for introducing priorities are reviewed.

transitions, and s_0 is the initial state. Each transition belonging to δ is a tuple $(s, i, o, \overline{p}, s')$ where $s, s' \in S$ are the initial and final states, $i \in I$ is an input action, $o \in O$ is an output action, and $\overline{p} \in \texttt{simbP}$ is the symbolic probability associated with the transition. We will usually denote transitions as $(s, i, o, \overline{p}, s')$ by $s \xrightarrow{i/o}_{\overline{p}} s'$. Besides, we consider that for any $s \in S$, $i \in I$, and the set $\alpha_{s,i} = \{s \xrightarrow{i/o}_{\overline{p}} s' \mid \exists\, o \in O, \overline{p} \in \texttt{simbP}, s' \in S : s \xrightarrow{i/o}_{\overline{p}} s' \in \delta\}$ the following two conditions hold:

- If $\mid \alpha_{s,i} \mid > 1$ then for any $s \xrightarrow{i/o}_{\overline{p}} s' \in \alpha_{s,i}$ we have that $1 \notin \overline{p}$.
- $1 \in \sum\{\overline{p} \mid \exists\, o \in O, s' \in S : s \xrightarrow{i/o}_{\overline{p}} s' \in \alpha_{s,i}\}$.

□

Intuitively, a transition $s \xrightarrow{i/o}_{\overline{p}} s'$ indicates that if the machine is in state s and receives the input i then, with a probability belonging to the interval $\overline{p}$, the machine emits the output o and evolves into s'. Let us comment the restrictions introduced at the end of the previous definition. The first constraint indicates that a symbolic probability such as $\overline{p} = \$p, 1]$ can appear in a transition $s \xrightarrow{i/o}_{\overline{p}} s' \in \delta$ only if it is the unique transition for s and i. Let us note that if there would exist two transitions $s \xrightarrow{i/o}_{\overline{p}} s', s \xrightarrow{i/o'}_{\overline{p}'} s'' \in \delta$ and the probability of one of them (say $\overline{p}$) included 1, then the probability associated to the other transition ($\overline{p}'$) could be 0, which is forbidden. Regarding the second condition, since the *real* probabilities for each state $s \in S$ and for each input $i \in I$ should add 1, then 1 must be within the lower and upper bounds of the associated symbolic probabilities.

Next we define some additional conditions that we will sometimes impose on our finite state machines.

Definition 3. Let $M = (S, I, O, \delta, s_0)$ be a PFSM. We say that M is *input-enabled* if for any $s \in S, i \in I$ there exist $s' \in S, o \in O, \overline{p} \in \texttt{simbP}$ such that $(s, i, o, \overline{p}, s') \in \delta$. We say that M is *deterministically observable* if for any $s \in S, i \in I, o \in O$ there do not exist two different transitions $(s, i, o, \overline{p}_1, s_1), (s, i, o, \overline{p}_2, s_2) \in \delta$. □

First, let us remark that the previous concepts are independent of the probabilistic information appearing in the state machines. Besides, the notion of deterministically observable is different from the more restricted notion of deterministic finite state machine. In particular, we allow transitions from the same state labeled by the same input action, as far as the outputs are different.

Example 1. Let us consider the (probabilistic) finite state machines depicted in Figure 1. For the sake of clarity we have not included probabilistic information in the graphs. Let us consider $M_3 = (\{1, 2, 3\}, \{i_1, i_2\}, \{o_1, o_2, o_3\}, \delta, 1)$. Next we define the set of transitions δ. For the first state we suppose the transitions $(1, i_1, o_1, 1, 2)$, $(1, i_2, o_1, \overline{p}_1, 1)$, and $(1, i_2, o_2, \overline{p}_2, 3)$, where $\overline{p}_1 = (0, \frac{1}{2}]$, $\overline{p}_2 = [\frac{1}{3}, 1)$, and let us remind that we denote the *interval* $[1, 1]$ simply by 1. We also know that the real probabilities associated with the last two transitions, say p_1 and p_2, are such that $p_1 + p_2 = 1$. A similar assignment of symbolic probabilities can be done to the rest of transitions appearing in the graph.

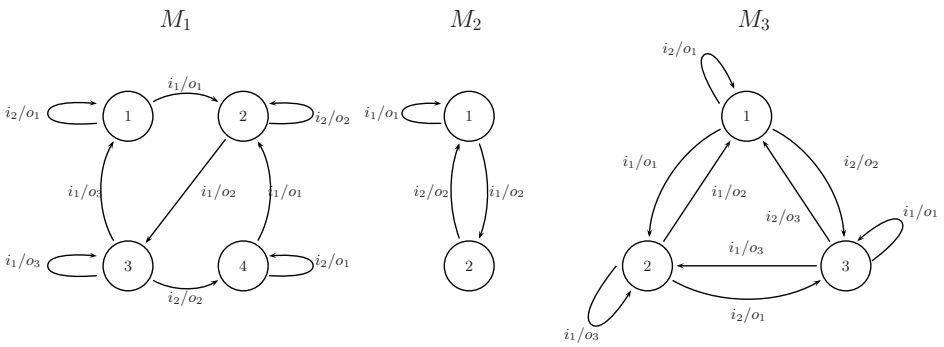

Fig. 1. Examples of PFSM.

Regarding the notions of input-enabling and deterministically observable, we have that M_1 fulfills the first of the properties but not the second one (there are two transitions outgoing from the state 3 labeled by i_1/o_3). The first property does not hold in M_2 (there is no outgoing transition labeled by i_2 from the state 2) while the second one does. Finally, M_3 holds both properties. □

As usually, we need to consider not only single evolutions of a PFSM but also sequences of transitions. Thus, we introduce the notion of (probabilistic) trace. We will associate probabilities to traces. The probability of a trace will be obtained by multiplying the probabilities of all transitions involved in the trace.

Definition 4. Let $M = (S, I, O, \delta, s_0)$ be a PFSM. We write the *generalized* transition $s \xRightarrow{(i_1/o_1,\ldots,i_n/o_n)}_{\overline{p}} s'$ if there exist $s_1, \ldots, s_{n-1} \in S, \overline{p}_1, \ldots, \overline{p}_n \in \texttt{simbP}$ such that $s \xrightarrow{i_1/o_1}_{\overline{p}_1} s_1 \xrightarrow{i_2/o_2}_{\overline{p}_2} s_2 \cdots s_{n-1} \xrightarrow{i_n/o_n}_{\overline{p}_n} s'$ and $\overline{p} = \prod \overline{p}_i$.

We say that $\rho = (i_1/o_1, \ldots, i_n/o_n)$ is a *non-probabilistic trace*, or simply a *trace*, of M if there exist $s' \in S$ and $\overline{p} \in \texttt{simbP}$ such that $s_0 \xRightarrow{\rho}_{\overline{p}} s'$.

Let $\rho = (i_1/o_1, \ldots, i_n/o_n)$ and $\overline{p} \in \texttt{simbP}$. We say that $\overline{\rho} = (\rho, \overline{p})$ is a *probabilistic trace* of M if there exists $s' \in S$ such that $s_0 \xRightarrow{\rho}_{\overline{p}} s'$.

We denote by $\texttt{Traces}(M)$ and $\texttt{pTraces}(M)$ the sets of non-probabilistic and probabilistic traces of M, respectively. □

We conclude this section by introducing notation related to hypothesis contrasts (an operational definition of these concepts will be given in the appendix of this paper). In the following definition we call *event* to any reaction we can detect from a system or environment. A *sample* contains information about the number of times we have detected each event along a set of observations. Besides, we associate a random variable with each set of events. Its purpose is to provide the theoretical (*a priori*) probability of each event in the set. In our framework, these random variables will be inferred from the PFSMs denoting the (ideal) probabilistic behavior of systems, while the samples will be collected by interacting with the

implementation under test. We will consider a variant of random variable allowing to deal with *symbolic* probabilities, as our PFSMs do. Besides, we provide a function that returns the confidence we have that a sample of events has been produced according to a given random variable. This function encapsulates the subjacent hypothesis contrast.

Definition 5. Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ be a set of *events*. A *sample* of $\mathcal{A}$ is a set $J = \{(a_1, m_1), \ldots, (a_n, m_n)\}$ where for any $1 \leq i \leq n$ we have that m_i represents the number of times that we have observed the event α_i.

Let $\xi : \mathcal{A} \to \texttt{simbP}$ be a function such that $1 \in \sum_{\alpha \in \mathcal{A}} \xi(\alpha)$. In this case we say that ξ is a *symbolic random variable* for the set of events $\mathcal{A}$. We denote the set of symbolic random variables for the set of events $\mathcal{A}$ by $\mathcal{RV}(\mathcal{A})$. We denote the set of symbolic random variables for any set of events by $\mathcal{RV}$.

Given the symbolic random variable ξ and the sample J we denote the *confidence* of ξ on J by $\gamma(\xi, J)$. □

We assume that $\gamma(\xi, J)$ takes values in the interval $[0, 1]$. Intuitively, bigger values of $\gamma(\xi, J)$ denote that the observed sample J is more likely to be produced by the symbolic random variable ξ. There exist several hypothesis contrasts to compute these confidence levels. In the appendix of this paper we show one of them to indicate how the notion of confidence may be formally defined.

In the next definition we particularize the previous notions in the context of our framework. Given a sequence of inputs we consider the sequence of input/outputs that the system can return. Hence, the set of events are those sequences of outputs that could be produced in response. The random variable to denote the theoretical probability of each event is computed by considering the symbolic probability of the corresponding trace in the specification.

Definition 6. Let $M = (S, I, O, \delta, s_0)$ be a PFSM and $\pi = (i_1, \ldots, i_n)$ be a sequence of inputs. The *set of trace events* associated to M with respect to π, denoted by $\texttt{TraceEvents}(M, \pi)$, is defined as

$$\texttt{TraceEvents}(M, \pi) = \left\{ (o_1, \ldots, o_n) \,\middle|\, (i_1/o_1, \ldots, i_n/o_n) \in \texttt{Traces}(M) \right\}$$

The *symbolic random variable* associated to the previous events, denoted by ξ^{π}_{M}, is defined in such a way that for any $(o_1, \ldots, o_n) \in \texttt{TraceEvents}(M, \pi)$ we have $\xi^{\pi}_{M}(o_1, \ldots, o_n) = \overline{p}$, being $((i_1/o_1, \ldots, i_n/o_n), \overline{p}) \in \texttt{pTraces}(M)$. □

3 Testing Probabilistic Systems

In this section we introduce the notion of test and we present how they are applied to implementations. In our context, to test an IUT consists in applying a sequence of inputs to the IUT. Once an output is received we check whether it is an expected one or not. In the former case, either a pass signal is emitted (indicating successful termination) or the testing process continues by applying another input. In the latter case, a fail signal is produced, the testing process stops, and we conclude that the implementation does not conform to the specification.

As we indicated in the introduction, the methodology to *guess* the probabilities associated with each bundle associated with an input action in the implementation consists in applying several times the same test. If we are testing an IUT with input and output sets I and O, respectively, tests are deterministic acyclic I/O labeled transition systems (i.e. trees) with a strict alternation between an input action and the whole set of output actions. A branch labeled by an output action can be followed by a leaf or by another input action. Moreover, leaves of the tree represent either *successful* or *failure* states. We will collect a sample of successes and failures of the test (one for each test execution) and, in the successful case, the sequences of input/output actions performed. With this information we will experimentally compute the probabilities associated with input actions. In addition, successful states will have a symbolic random variable associated with them. This random variable will denote the *probabilistic constraint* imposed in the test for the trace leading to that state. Basically, a hypothesis contrast will compare the samples collected for that event with the probabilistic constraint imposed by the test.

Definition 7. A *test* is a tuple $T = (S, I, O, \delta, s_0, S_I, S_O, S_F, S_P, \zeta)$ where S is the set of states, I and O, with $I \cap O = \emptyset$, are the sets of input and output actions, respectively, $\delta \subseteq S \times I \cup O \times S$ 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 any input state $s \in S_I$ there exists a unique outgoing transition $(s, i, s') \in \delta$. For this transition we have that $i \in I$ and $s' \in S_O$.
- S_O is the set of *output* states. For any output state $s \in S_O$ we have that for any $o \in O$ there exists a unique state $s' \in S$ such that $(s, o, s') \in \delta$; in each case, $s' \notin S_O$. Moreover, there do not exist $i \in I$ and $s' \in S$ such that $(s, i, s') \in \delta$.
- S_F and S_P are the sets of *fail* and *pass* states, respectively. We say that these states are *terminal*. That is, for any 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 $(s, a, s') \in \delta$.

Finally, $\zeta : S_P \longrightarrow \mathcal{RV}$ is a function associating passing states with (symbolic) random variables.

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

Next we define the set of traces that a test can perform. These traces are sequences of input/output actions reaching terminal states. Depending on the final state we will classify them as either *successful* or *failure* traces.

Definition 8. Let $\rho = (i_1/o_1, \ldots, i_r/o_r)$ be a sequence of input/output actions, $T = (S, I, O, \delta, s_0, S_I, S_O, S_F, S_P, \zeta)$ be a test, and $s \in S$. We say that ρ is a *trace of T reaching s*, denoted by $T \stackrel{\rho}{\Longrightarrow} s$, if $s \in S_F \cup S_P$ and there exist states $s_{12}, s_{21}, s_{22}, \ldots s_{r1}, s_{r2} \in S$ such that $\{(s_0, i_1, s_{12}), (s_{r2}, o_r, s)\} \subseteq \delta$, and for any $2 \leq j \leq r$ we have $(s_{j1}, i_j, s_{j2}) \in \delta$ and $(s_{(j-1)2}, o_{j-1}, s_{j1}) \in \delta$. □

The next definition presents some auxiliary predicates that we will use during the rest of the paper. While the first two notions are easy to understand, the last one needs some additional explanation. Given a trace ρ and a set H of pairs (trace,natural number), $\texttt{IPrefix}(H, \rho)$ is another set of pairs including all traces such that its sequence of input actions matches that of ρ. The number attached to each trace corresponds with the number of traces belonging to H *beginning* with that trace. For instance, let us consider the set of pairs $H = \{((i_1/o_1, i_2/o_1), 1), ((i_1/o_2, i_1/o_2), 2), ((i_1/o_2, i_2/o_1), 3), ((i_2/o_1, i_2/o_2), 4)\}$ and let us apply the function to the trace (i_1/o_1). Then, the resulting set is $H' = \{((i_1/o_1), 1), ((i_1/o_2), 5)\}$. Let us remark that we discard the outputs of the trace considered as parameter. For example, the set H' would be the same if we consider the trace (i_1/o_2) instead of the trace (i_1/o_1). Given a sample of executions from an implementation, we will use this function to compute the number of times that the implementation has performed each sequence of outputs in response to some sequence of inputs. Let us note that if we observe that the sequence of outputs $(o_1, \ldots, o_n)$ has been produced in response to the sequence of inputs $(i_1, \ldots, i_n)$ then, for any $j \leq n$, we know that the sequence of outputs $(o_1, \ldots, o_j)$ has been produced in response to $(i_1, \ldots, i_j)$. Hence, the observation of a trace is useful to compute the number of instances of its prefixes.

Definition 9. Let $\sigma = (u_1, \ldots, u_n)$ and $\sigma' = (u'_1, \ldots, u'_m)$ be two sequences. We say that σ is *a prefix of* σ', denoted by $\texttt{Prefix}(\sigma, \sigma')$, if $n < m$ and for any $1 \leq i \leq n$ we have $u_i = u'_i$.

Let $\rho = (i_1/o_1, \ldots, i_m/o_m)$ be a sequence of input/output actions. We define the *input actions of the sequence* ρ, denoted by $\texttt{inputs}(\rho)$, as the sequence $(i_1, \ldots, i_m)$, and the *output actions of the sequence* ρ, denoted by $\texttt{outputs}(\rho)$, as the sequence $(o_1, \ldots, o_m)$.

Let $H = \{(\rho_1, r_1), \ldots, (\rho_m, r_m)\}$ be a set of pairs (trace, natural number) and $\rho = (i_1/o_1, \ldots, i_n/o_n)$ be a trace. The *set of input prefixes* of ρ in H, denoted by $\texttt{IPrefix}(H, \rho)$, is defined as

$$\texttt{IPrefix}(H, \rho) = \left\{ (\rho', r') \;\middle|\; \begin{array}{l} \texttt{inputs}(\rho) = \texttt{inputs}(\rho') \ \wedge \ r' > 0 \ \wedge \\ r' = \sum \{\!\!\{ r'' \mid (\rho'', r'') \in H \ \wedge \ \texttt{Prefix}(\rho', \rho'') \}\!\!\} \end{array} \right\}$$

□

Next we present the notions that we will use to denote that a given event has been detected in an IUT. We will also compute the sequences of actions that the implementation performs when a test is applied.

Definition 10. Let $\mathcal{I} = (S, I, O, \delta, s_0)$ be a PFSM representing an IUT. We say that $(i_1/o_1, \ldots, i_n/o_n)$ is an *execution* of $\mathcal{I}$ if the sequence $(i_1/o_1, \ldots, i_n/o_n)$ can be performed by $\mathcal{I}$.

Let $\rho_1, \ldots, \rho_n$ be executions of $\mathcal{I}$ and $r_1, \ldots, r_n \in \mathbb{N}$. We say that the set $H = \{(\rho_1, r_1), \ldots, (\rho_n, r_n)\}$ is an *execution sample* of $\mathcal{I}$.

Let $T = (S', I, O, \delta', s'_0, S_I, S_O, S_F, S_P, \zeta)$ be a valid test. We say that $H = \{(\rho_1, r_1), \ldots, (\rho_n, r_n)\}$ is an *execution sample of* $\mathcal{I}$ *under a test* T if H is an execution sample and for any $(\rho, r) \in H$ we have that $T \stackrel{\rho}{\Longrightarrow} s$, with $s \in S'$.

Let $\Omega = \{T_1, \ldots, T_n\}$ be a set of tests and let $H_1, \ldots, H_n$ be execution samples of $\mathcal{I}$ under T_i. We say that $H = \{(\rho_1, r_1), \ldots, (\rho_n, r_n)\}$ is an *execution sample of $\mathcal{I}$ under the test suite Ω* if for any $(\rho, r) \in H$ we have that r is given by the expression $r = \sum\{\!|r' \mid 1 \leq i \leq n \ \wedge \ (\rho, r') \in H_i|\!\}$. □

In the definition of execution sample under a test we have that each number r, with $(\rho, r) \in H$, denotes the number of times we have observed the execution ρ in $\mathcal{I}$ under the (repeated) application of T.

Now we present the conditions required to *pass* a test. Passing a test consists in fulfilling two different constraints. First, we require that the test never reaches a failure state as a result of its interaction with the implementation. This condition concerns what is *possible*. Second, we require that the random variables attached to successful states conform to the samples collected during the (repeated) application of the test to the IUT. This condition concerns what is *probable*. We will consider that the set of executions analyzed to pass a test does not only include those executions obtained by applying that test, but also the executions obtained by applying other tests. Let us remark that the very same traces that are available in a test could be part of other tests as well. Let us also note that the validity of any hypothesis contrast improves with the number of samples. Hence, it would not be efficient to apply each hypothesis contrast to the *limited* collection of samples obtained by a single test. On the contrary, samples collected by different tests will be shared so that our statistical information grows and the hypothesis contrast procedure improves. Let us note that this testing methodology is opposite to usual techniques where the application of each test is independent from other tests.

Definition 11. Let $H = \{(\rho_1, r_1), \ldots, (\rho_n, r_n)\}$ be an execution sample of $\mathcal{I}$ under the test suite $\Omega = \{T_1, \ldots, T_n\}$ and let $0 \leq \alpha \leq 1$. Let us consider $T \in \Omega$. We say that the implementation $\mathcal{I}$ $(\alpha, H)-$*passes* the test T if for any trace $\rho \in \mathtt{Traces}(\mathcal{I})$, with $T \stackrel{\rho}{\Longrightarrow} s$, we have that $s \notin S_F$ and if $s \in S_P$ then $\gamma(\zeta(s), R) > \alpha$, where

$$R = \{(\mathtt{outputs}(\rho'), r) \mid (\rho', r) \in \mathtt{IPrefix}(H, \rho)\}$$

We say that $\mathcal{I}$ $(\alpha, H)-$*passes* the test suite Ω if $\mathcal{I}$ $(\alpha, H)-$*passes* T_i, for any $1 \leq i \leq n$. □

4 Implementation Relations based on Samples

In this section we introduce two new implementation relations that take into account the practical limitations to collect probabilistic information from an implementation. These relations allow us to claim the accurateness of the probabilistic behavior of an implementation with respect to a specification *up to* a given confidence level. Given a set of execution samples, we will apply a hypothesis contrast to check whether the probabilistic choices taken by the implementation follow the patterns given by the specification.

Our implementation relations follow the classical pattern of formal conformance relations defined in systems distinguishing between inputs and outputs (see e.g. [17,18]). That is, an IUT conforms to a specification $\mathcal{S}$ if for any possible

evolution of $\mathcal{S}$ the outputs that the IUT may perform after a given input are a subset of those for the specification. Let us remark that this constraint could be rewritten in probabilistic terms: The confidence we have on the fact that the implementation will not perform forbidden behaviors is 1 (i.e. *complete*). However, since no hypothesis contrast can provide full confidence, it is preferable to keep the constraints over actions separated from the probabilistic constraints and deal with them in the classic way, that is, an implementation is incorrect with respect to forbidden behavior if such a behavior is detected. Let us remind that the reverse is not true: We cannot claim that the implementation is correct even if no forbidden behavior is detected after a finite number of interactions with it.

Definition 12. Let $\mathcal{S}$ and $\mathcal{I}$ be PFSMs. We say that $\mathcal{I}$ *non-probabilistically conforms* to $\mathcal{S}$, denoted by $\mathcal{I}$ $\mathtt{conf}$ $\mathcal{S}$, if for any $\rho = (i_1/o_1, \ldots, i_n/o_n) \in \mathtt{Traces}(\mathcal{S})$, with $n \geq 1$, we have

$$\rho' = (i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n/o'_n) \in \mathtt{Traces}(\mathcal{I}) \text{ implies } \rho' \in \mathtt{Traces}(\mathcal{S})$$

□

Regarding the probabilistic constraints of the specification, in both relations we put together all the observations of the implementation. Then, the set of samples corresponding to each trace of the specification will be composed by taking all the observations such that the trace is a *prefix* of them. By doing so we will be able to compare the number of times the implementation has performed the chosen trace with the number of times the implementation has performed any other behavior. We will use hypothesis contrasts to decide whether the probabilistic choices of the implementation conform to the probabilistic constraints imposed by the specification. In particular, a hypothesis contrast will be applied to each sequence of inputs considered by the specification. This contrast will check whether the different sequences of outputs associated with these inputs are distributed according to the probability distribution of the random variable associated with that sequence of inputs in the specification. In the next definition we introduce our first implementation relation based on samples.

Definition 13. Let $\mathcal{S}$ be a specification and $\mathcal{I}$ be an IUT. Let H be an execution sample and let $0 \leq \alpha \leq 1$. We say that $\mathcal{I}$ $(\alpha, H)-$*probabilistically conforms to* $\mathcal{S}$, denoted by $\mathcal{I}$ $\mathtt{confp}^{(\alpha,H)}$ $\mathcal{S}$, if $\mathcal{I}$ $\mathtt{conf}$ $\mathcal{S}$ and for any $\rho \in \mathtt{Traces}(\mathcal{S})$ we have $\gamma(\xi^{\pi}_{\mathcal{S}}, R) > \alpha$, where $\pi = \mathtt{inputs}(\rho)$ and

$$R = \{(\mathtt{outputs}(\rho'), r) \mid (\rho', r) \in \mathtt{IPrefix}(H, \rho)\}$$

□

In the previous relation $\xi^{\pi}_{\mathcal{S}}$ denotes the symbolic random variable associated with the sequence of input actions π for the PFSM $\mathcal{S}$ (see Definition 6). Besides, each trace observed in the implementation will add one instance to the accounting of its prefixes. We could consider an alternative procedure where traces are independently accounted and each observed trace does not affect the number of instances of other traces being prefix of it. However, this method would lose valuable

information that might negatively affect the quality of the hypothesis contrasts. Let us remind that the reliability of any hypothesis contrasts increases with the number of instances included in the samples. Besides, as we said before, an observation where $(o_1, \ldots, o_n)$ has been produced in response to $(i_1, \ldots, i_n)$ is indeed an observation where, in particular, $(o_1, \ldots, o_j)$ has been produced in response to $(i_1, \ldots, i_j)$, with $j \leq n$. So, by accounting prefixes we properly increase the number of instances processed by hypothesis contrasts, which makes them more precise (as well as the probabilistic implementation relation that takes them into account).

The previous idea induces the definition of a new implementation relation that is a refinement of the previous one. Let us note that the probability of observing a given trace decreases, in general, as the length of the trace increases. This is so because more probabilistic choices are taken in long traces. Besides, taking prefixes into account increases the number of instances of *short* traces. Thus, it is likely that the number of short traces applied to the hypothesis contrasts of the previous relation will outnumber that of longer traces. Let us note that statistical noise effects are higher when smaller sets of samples are considered. Moreover, if we consider extremely long traces we could obtain a couple of instances or even none in each class of events to be considered by a hypothesis contrast, which would ruin the result of such contrast. Taking these factors into account, in the next definition we introduce a new implementation relation where the confidence requirement is *relaxed* as the length of the trace growths. This reduction is defined by a non-increasing function associating confidence levels to the length of traces.

Definition 14. Let $f : \mathbb{N} \to \mathbb{R}^+$ be a strictly non-increasing function, $\mathcal{S}$ be a specification, and $\mathcal{I}$ be an implementation. Let H be an execution sample of $\mathcal{I}$. We say that $\mathcal{I}$ $(f, H)-$*probabilistically conforms to* $\mathcal{S}$, denoted by $\mathcal{I}\,\mathtt{confp}^{(f,H)}\,\mathcal{S}$, if $\mathcal{I}\,\mathtt{conf}\,\mathcal{S}$ and for any $\rho \in \mathtt{Traces}(\mathcal{S})$ we have $\gamma(\xi^{\pi}_{\mathcal{S}}, R) > f(l)$, where $\pi = \mathtt{inputs}(\rho)$, l is the length of $\mathtt{inputs}(\rho)$, and

$$R = \{(\mathtt{outputs}(\rho'), r) \mid (\rho', r) \in \mathtt{IPrefix}(H, \rho)\}$$

□

5 Test Derivation

In this section we provide an algorithm to derive tests from specifications. In addition, we will show that the derived test suite is *complete* up to a given confidence α with respect to the conformance relation presented in Definition 13. As usually, the idea consists in traversing the specification to get all the possible traces in the adequate way. Thus, each test is generated so that it *focuses* on chasing a concrete trace of the specification. The test cases will contain probabilistic constraints so that they can detect faulty probabilistic behaviors in the IUT. Thus, successful states will have attached symbolic random variables that impose some constraints concerning the trace executed so far in the test. First, we give some auxiliary functions.

Input: $M = (S, I, O, \delta, s_0)$.
Output: $T = (S', I, O, \delta', s'_0, S_I, S_O, S_F, S_P, \zeta)$.

Initialization:

- $S' := \{s'_0\}, \delta' := S_I := S_O := S_F := S_P := \emptyset$.
- $S_{aux} := \{(s_0, s'_0, (\,))\}$.

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

1. If $(s^M, s^T, \pi) \in S_{aux}$ then perform the following steps:
   (a) $S_{aux} := S_{aux} - \{(s^M, s^T, \pi)\}$.
   (b) $S_P := S_P \cup \{s^T\}$.
   (c) $\zeta(s^T) := \xi^{\pi}_M$, where $\pi = \pi' \circ i$.
2. If $S_{aux} = \{(s^M, s^T, \pi)\}$ is a unitary set and there exists $i \in I$ such that $\mathtt{out}(s^M, i) \neq \emptyset$ then perform the following steps:
   (a) $S_{aux} := \emptyset$.
   (b) Choose i such that $\mathtt{out}(s^M, i) \neq \emptyset$.
   (c) Create a fresh state $s' \notin S'$ and perform $S' := S' \cup \{s'\}$.
   (d) $S_I := S_I \cup \{s^T\}$; $S_O := S_O \cup \{s'\}$; $\delta' := \delta' \cup \{(s^T, i, s')\}$.
   (e) For each $o \notin \mathtt{out}(s^M, i)$ do
      - Create a fresh state $s'' \notin S'$ and perform $S' := S' \cup \{s''\}$.
      - $S_F := S_F \cup \{s''\}$; $\delta' := \delta' \cup \{(s', o, s'')\}$.
   (f) For each $o \in \mathtt{out}(s^M, i)$ do
      - Create a fresh state $s'' \notin S'$ and perform $S' := S' \cup \{s''\}$.
      - $\delta' := \delta' \cup \{(s', o, s'')\}$.
      - $s_1^M := \mathtt{after}(s^M, i, o)$.
      - Let $(s^M, i, o, \overline{p}, s_1^M) \in \delta$. $S_{aux} := S_{aux} \cup \{(s_1^M, s'', \pi \circ i)\}$.

Fig. 2. Test Derivation Algorithm.

Definition 15. Let $M = (S, I, O, \delta, s_0)$ be a PFSM. We define the set of possible outputs in state s after input i as $\mathtt{out}(s, i) = \{o \mid \exists s' : (s, i, o, \overline{p}, s') \in \delta\}$. For any transition $(s, i, o, \overline{p}, s') \in \delta$ we write $\mathtt{after}(s, i, o) = s'$. □

Let us remark that, due to the assumption that PFSMs are deterministically observable, $\mathtt{after}(s, i, o)$ is uniquely determined.

Our derivation algorithm is presented in Figure 2. By considering the possible available choices we get a set of tests extracted from the specification M. We denote this set of tests by $\mathtt{tests}(M)$. In this algorithm, the set of pending states S_{aux} keeps track of the states of the test whose definition has not been *finished* yet. A tuple $(s^M, s^T, \pi) \in S_{aux}$ indicates that the current state in the traversal of the specification is s^M, that we did not conclude yet the description of the state s^T in the test, and that the sequence of inputs traversed from s_0 to s^T is π. The set S_{aux} initially contains a tuple with the initial states (of both specification and test) and an empty sequence of inputs. For each tuple in S_{aux} we may choose one possibility. It is important to remark that the second possibility is applied at most

to one of the possible tuples. Thus, our derived tests correspond to valid tests as introduced in Definition 7.

The first possibility simply indicates that the state of the test becomes a successful state. In this case, we attach a symbolic random variable to this state. This random variable must encode the probability distribution, according to the specification, for all possible traces having the sequence of inputs π.

The second possibility takes an input and generates a transition in the test labeled by this input. Then, the whole sets of outputs is considered. If the output is not expected by the implementation then a transition leading to a failure state is created. This could be simulated by a single branch in the test, labeled by `else`, leading to a failure state (in the algorithm we suppose that *all* the possible outputs appear in the test). For the rest of outputs we create a transition with the corresponding output 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.

The next results states that, for any specification $\mathcal{S}$, the test suite $\texttt{tests}(\mathcal{S})$ can be used to distinguish those (and only those) implementations that conform with respect to `confp`. However, we cannot properly say that the test suite is complete since both passing tests and the considered implementation relation have a probabilistic component. So, we can speak about *completeness* up to a certain confidence level. The proof of the *non-probabilistic* part of the result is strongly based on that for `ioco` [17]. The proof of the *probabilistic* component follows the scheme introduced for the relation `confs` [14] (a complete proof can be found in [16]).

Proposition 1. Let $\mathcal{I}$ and $\mathcal{S}$ be `PFSM`s. For any $0 \leq \alpha \leq 1$ and execution sample H we have $\mathcal{I}\ \texttt{confp}^{(\alpha,H)^{tr}}\ \mathcal{S}$ iff $\mathcal{I}$ $(\alpha, H)-$*passes* $\texttt{tests}(\mathcal{S})$. □

6 Conclusions and Future Work

In this paper we have presented a testing methodology to check whether an implementation properly follows the behavior described by a given specification. The particularity of our framework is that specifications can explicitly express the desired *propensity* of each option in each non-deterministic choice of the system. This propensity is denoted in terms of probabilities. Moreover, in order to improve the expressivity of specifications, symbolic probabilities are introduced. These features increase the complexity of the testing methodology, as it is impossible to infer the actual probabilities associated with implementations from a set of interaction samples. In order to cope with this problem, hypothesis contrasts are used.

As future work we plan to study the integration of this framework within that presented in [14], where a testing methodology for stochastic timed processes is introduced. Moreover, we are already implementing a tool to apply both the testing framework presented in this paper and the one given in [14].

References

1. J.C.M. Baeten and C.A. Middelburg. *Process algebra with timing*. EATCS Monograph. Springer, 2002.
2. K.A. Bartlett, R.A. Scantlebury, and P.T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.
3. B. Beizer. *Black Box Testing*. John Wiley and Sons, 1995.
4. 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.
5. R. Cleaveland, G. Lüttgen, and V. Natarajan. Priority in process algebra. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of process algebra*, chapter 12. North Holland, 2001.
6. 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.
7. B. Jonsson, W. Yi, and K.G. Larsen. Probabilistic extensions of process algebras. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of process algebra*, chapter 11. North Holland, 2001.
8. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
9. N. López and M. Núñez. An overview of probabilistic process algebras and their equivalences. In *Validation of Stochastic Systems, LNCS 2925*, pages 89–123. Springer, 2004. To appear.
10. N. López, M. Núñez, and I. Rodríguez. Formal specification of symbolic-probabilistic systems. In *European Performance Engineering Workshop (EPEW'04), LNCS*. Springer, 2004. In press.
11. G.J. Myers. *The Art of Software Testing*. John Wiley and Sons, 1979.
12. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Computer Aided Verification'91, LNCS 575*, pages 376–398, 1991.
13. M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.
14. M. Núñez and I. Rodríguez. Towards testing stochastic timed systems. In *FORTE 2003, LNCS 2767*, pages 335–350. Springer, 2003.
15. G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.
16. I. Rodríguez. *Especificación de sistemas concurrentes usando conceptos de teoría económica: Sintaxis, semántica, aplicaciones y extensiones del lenguaje formal PAMR*. PhD thesis, Universidad Complutense de Madrid, 2004.
17. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.
18. J. Tretmans. Testing concurrent systems: A formal approach. In *CONCUR'99, LNCS 1664*, pages 46–65. Springer, 1999.
19. W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Protocol Specification, Testing and Verification XII*, pages 47–61. North Holland, 1992.

Appendix. Statistics Background: Hypothesis Contrasts

This appendix is given for the reviewers and for the sake of completeness. It has been extracted from [10].

In this appendix we introduce one of the standard ways to measure the confidence that a random variable has on a sample. In order to do so we will present a methodology to perform *hypothesis contrasts*. Intuitively, 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 that we have detected 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 n we perform the following steps:

- We split the sample into k classes covering all the possible range of values. We denote by O_i the *observed frequency* in class i (i.e. the number of elements belonging to the class i).
- We calculate, according to the proposed random variable, the probability p_i of each class i. We denote by E_i the *expected frequency* of class i, that is, $E_i = np_i$.
- We calculate the *discrepancy* between observed and expected frequencies as $X^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$. When the model is correct, this discrepancy is approximately distributed as a random variable χ^2.
- The number of freedom degrees of χ^2 is $k - 1$.
- We will *accept* that the sample follows the proposed random variable if the probability to obtain a discrepancy greater than or equal to the detected discrepancy is high enough, that is, if $X^2 < \chi^2_\alpha(k-1)$ for some α high enough. Actually, as such margin to accept the sample decreases as α increases, we can obtain a measure of the validity of the sample as $\max\{\alpha \mid X^2 \leq \chi^2_\alpha(k-1)\}$.

According to the previous steps, we can now present an operative definition of the function γ which has been presented before in Definition 5. Since we will use hypothesis contrasts to compare samples with *symbolic* random variables but the previous procedure refers to *standard* random variables, we must be carefull when applying the previous ideas in our framework. Let us note that symbolic random variables encapsulate a set of standard random variables (this set is in general infinite). For instance, let us consider the set of events $\mathcal{A} = \{a, b\}$ and the symbolic random variable $\xi : \mathcal{A} \to \texttt{simbP}$ with $\xi(a) = \xi(b) = (\frac{1}{4}, \frac{3}{4})$. Then, a possible standard random variable fitting into ξ is $\xi' : \mathcal{A} \to (0, 1]$ with $\xi'(a) = \frac{1}{3}$ and $\xi'(b) = \frac{2}{3}$. Another possibility is $\xi'' : \mathcal{A} \to (0, 1]$ with $\xi''(a) = \xi''(b) = \frac{1}{2}$. Since ξ embraces both possibilities, assessing the confidence of ξ on a sample should consider both of them. Actually, we will consider that the sample is adequate for ξ if it would be so for some standard random variable fitting into ξ. More generally, an *instance* of a symbolic random variable is a (standard) random variable where each probability fits into the margins of the symbolic random variable for the corresponding class. Besides, the addition of the probabilities must be equal to 1. In order to compute the confidence of a symbolic random variable on a sample we consider the instance of it that returns the highest confidence on that sample.

Definition 16. Let $\mathcal{A} = \{a_1, \ldots, a_k\}$ be a set of events, $\xi : \mathcal{A} \to \texttt{simbP}$ be a symbolic random variable, $\xi' : \mathcal{A} \to (0, 1]$ be a random variable, and J be a sample of $\mathcal{A}$. We say that the random variable ξ' is an *instantiation* of ξ, denoted by $\texttt{Instance}(\xi', \xi)$, if for any $a \in \mathcal{A}$ we have $\xi'(a) \in \xi(a)$ and $\sum_{a \in \mathcal{A}} \xi'(a) = 1$.

For any random variable $\xi' : \mathcal{A} \to (0, 1]$ let X^2 denote the discrepancy level of J on ξ' calculated as explained above by splitting the sampling space into the set of events $\mathcal{A}$. Let $\xi : \mathcal{A} \to \texttt{simbP}$ denote a symbolic random variable. We define the confidence of ξ on J, denoted by $\gamma(\xi, J)$, as follows:

$$\gamma(\xi, J) = \max \left\{ \alpha \;\middle|\; \begin{array}{l} \exists\, \xi' : \texttt{Instance}(\xi', \xi) \wedge \\ \qquad \alpha = \max\{\alpha' \mid X^2 \leq \chi^2_{\alpha'}(k-1)\} \end{array} \right\}$$

□