Derivation of a Suitable Finite Test Suite for Customized Probabilistic Systems *

Luis F. Llana-Díaz, Manuel Núñez, and Ismael Rodríguez

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

Abstract. In order to check the conformance of an IUT (implementation under test) with respect to a specification, it is not feasible, in general, to test the whole set of IUT available behaviors. In some situations, testing the behavior of the IUT assuming that it is stimulated by a given usage model is more appropriate. Specifically, if we consider that specifications and usage models are defined in probabilistic terms, then by applying a finite set of tests to the IUT we can compute a relevant metric: An upper bound of the probability that a user following the usage model finds an error in the IUT. Our previous work allows to assess this metric by considering, on the one hand, the whole set of traces of the composition of the specification and the user model and, on the other hand, the behavior of the IUT. However, a more precise metric can be obtained if we separately analyze the behavior of the IUT for each sequence of inputs and then we appropriately combine the results. In this paper we extend the methodology to follow this approach. In addition, we use the method to find an *optimal* (with respect to the number of inputs) set of tests that minimizes that upper bound.

1 Introduction

In order to test the behavior of an IUT (*implementation under test*) sometimes it is preferable to check only some functionalities that are specially relevant or critical. In this line, we can consider that the IUT is analyzed in the context of a specific usage model or, more generally, in terms of its interaction with a (probably abstract) *user* that represents some manners to interact with the IUT. Let us note that if only the functional behavior of systems is considered (that is, we just check what must or must not be done), then this kind of *user-customized* approach consists in testing a *subset* of the behaviors defined by the specification. However, if other kinds of features are taken into account then this approach might provide some interesting possibilities. In particular, if specifications and *user models* are defined in *probabilistic* terms then we can calculate a measure that cannot be computed otherwise: After a finite set of tests is applied to the IUT, we can calculate a measure of the probability that a user behaving according to the user model finds a *wrong* behavior in the IUT. That is, after a finite subset of the infinite set of relevant behaviors is analyzed, we will be provided with a *global* measure of correctness of the IUT.

We can do it as follows. First, we choose some tests that exercise some behaviors concerned by the user model. Then, we apply the tests to the IUT to check whether

* Research partially supported by the Spanish MCYT project TIC2003-07848-C02-01, the Junta de Castilla-La Mancha project PAC-03-001, and the Marie Curie project MRTN-CT-2003-505121/TAROT.

the behaviors exercised by these tests are *correct* with respect to the specification. Basically, tests induce some stimuli (sequences of inputs) and we observe the response (sequences of outputs) produced by the IUT. Since specifications are defined in probabilistic terms, the correctness of the IUT must be assessed in these terms as well. However, we cannot *read* the probabilities associated to a probabilistic choice of the IUT since we consider it to be a *black box*. For instance, if the IUT produces a with probability 0.5 then we can observe that a is produced or not, but the value 0.5 is not visible. In fact, our process to assess the IUT will be probabilistic. For each specific *behavior case* (i.e., sequence of inputs) analyzed by tests, we apply a *hypothesis contrast* to check whether we can claim, for a given level of *confidence* α, that the answers (i.e., sequences of outputs) produced by the IUT behave as required by the specification. On the one hand, the specification will define the probability of each choice by means of a *random variable*. On the other hand, a sample of the IUT will be collected by means of interacting with it. Then, the hypothesis contrast checks whether, for a given confidence α, we can claim that the IUT sample is produced accordingly to the specification random variable. For instance, if the specification says that a and b are produced with 0.5 probability each, then the confidence will be high if they are observed 507 and 493 times, respectively. However, if they are produced 614 and 386 times then the confidence will be lower.

Let us suppose that, after a suitable hypothesis contrast is applied, a given IUT behavior case is *validated* with confidence α. Then we can assume, with that confidence, that the probability of each IUT response (in that behavior case) is actually defined as in the specification. Since user models and specifications define their activities in probabilistic terms, we can compute the probability that any sequence of inputs/outputs is produced when both models interact together. Let us remark that the probabilities governing the behavior of the IUT are actually unknown. However, we can assume that all behaviors that pass the hypothesis contrast behave as the specification defines with confidence α. Hence, under this confidence, we can calculate the probability that a sequence of inputs/outputs *whose behavior was validated* is produced during the interaction of the IUT and the user model. By using this information we will calculate our correctness metric: An *upper bound* of the probability that a wrong probabilistic behavior is observed when the user model and the IUT interact. In the *worst* case all behaviors that have not been either validated or observed are wrong. The probability of executing any of them (they are infinite) is the complementary of the probability of executing a validated one (they are finite and their probabilities are known). Hence, after a finite test suite is applied, we can compute the probability of taking a non-validated behavior, which is incorrect in the worst case. That is, we obtain an upper bound of the probability of finding an error in the IUT (with confidence α).

Let us note that this measure cannot be computed in other testing frameworks. In particular, if the probabilistic behavior of systems is not considered (or it is, but user models are not regarded), then after a finite test suite is applied the coverage of all relevant cases is null in general: Among an infinite set of behavior cases to be analyzed, an infinitesimal part of them are tested. Thus, without any additional assumptions, nothing about the correctness of this IUT can be claimed. Though the capability of a finite test suite to detect errors in an IUT can be assessed, it is done either in heuristic terms [8, 10, 2] or by adding some hypotheses about the IUT behavior that allow to reduce the number of cases to be tested [1, 11, 22, 17, 18]. On the contrary, our metric provides a (probabilistic) non-heuristic correctness measure without reducing the *testing space*.

In this paper we continue a previous work where the application of user models to compute a probabilistic measure of correctness of the IUT was proposed [12]. In that approach, the behavior of the composition of the IUT and the user model is probabilistically compared to the behavior of the composition of the specification and the user model. A *single* hypothesis contrast is used to compare the former (denoted by means of a single random variable) and the latter (represented by a sample denoting all observations). On the one hand, the random variable denotes the probability of *all* traces in the composition of the specification and the user model. On the other hand, the sample represents all traces observed in the IUT. Though this approach is simple and elegant, the precision of the analysis can be strongly improved if the events to be analyzed are further *decomposed*. In fact, even if we assume that user models are correct by definition, their probabilistic behavior can be a source of *sampling noise*. For instance, let us suppose that the user model chooses between a and b with equal probability. Then, in both cases, the specification answers c and d, again with equal probabilities. Though it is not very probable, the interaction of the user model and a *correct* IUT could produce a/c 100 times and b/c 100 times. In this case, we would (wrongly) deduce that the IUT is incorrect, because the specified probability of both a/c and b/c is $0.5 \cdot 0.5 = 0.25 \not\approx \frac{100}{200}$. Similarly, a/c and a/d could be produced 100 times each, which again is not accepted. However, this time the sample is rejected due to a (rare) behavior of the user model. In fact, assessing a random source (in our case, the IUT behavior) on the basis of its interaction with *another* random source that is correct (the user model) is less precise than assessing the random source by its own.[1] On the contrary, as we sketched before, in this paper we will apply a hypothesis contrast for each *behavior case* (i.e., sequence of inputs). Each hypothesis contrast application checks whether responses (i.e., sequences of outputs) are properly given for the considered sequence of inputs. In the previous example, the correctness of the IUT when a or b are produced is independently checked. Hence, the sample cannot be ruined by a rare behavior of the user model (of course, it can still be ruined by a rare behavior of a correct IUT). Our way to handle samples and tests will dramatically differ with respect to the previous approach. For each sequence of inputs, different tests will collect the answers (i.e. sequences of outputs) produced by the IUT in response to it. Besides, several random variables (one for each sequence of inputs), rather than a single variable, will be extracted from the specification. Hence, the development will be somehow opposite to that presented in [12]. Besides, in this paper we will use the metric of error probability of the IUT to find *optimal* sets of tests. Let us note that our metric does not only provide a correctness measure, but it also can be used to guide the testing process. In particular, given a user model and a specification, it allows to choose suitable finite sets of tests: If we compute that *passing* a given test suite Ω_1 would provide an error measure 0.3, while passing another suite Ω_2 would provide an error measure 0.2, then the suite Ω_2 is preferable. Moreover, given the number of inputs we wish to apply during the testing process, we can compute the *optimal* test suite such that the sum of lengths of all tests in the suite is under this threshold.

In terms of other related work, there is significant work on testing preorders and equivalences for probabilistic processes [4, 16, 20, 5, 3, 21, 14, 13]. Most of these proposals follow the *de Nicola and Hennessy's style* [6, 9], that is, two processes are equivalent if the application of any test belonging to a given set returns the same result. Instead, we are interested in checking whether an implementation conforms to a specification.

[1] Similarly, if we want to check whether a dice is *fair*, drawing it together with a fair dice and assessing the *sum* of results is less precise than drawing the dice isolated.

In particular, our relations are similar to the ones introduced in [22, 15]. Regarding probabilistic user models, it is worth to point out that these previous works do not explicitly consider this notion. User models have been used in specific software testing scenarios (e.g., to test C++ templates [19]). Other work deals with user models in the testing context [25, 24], but they do not consider formal conformance testing techniques. Besides, none of these works proposes a method to compute the probability of error of the IUT after a finite suite is applied, neither they allow to find optimal finite test suites.

The rest of the paper is structured as follows. In the next section we present some basic notions of our framework such as random variables, probabilistic finite state machines, and probabilistic labelled transitions systems. In Section 3 we introduce tests and we define the behavior of machines and users when they interact each other. In Section 4 we present implementation relations that allow to relate specifications and implementations *with respect to* a given user model. Next, in Section 5 we describe our method to calculate an upper bound of the error probability of an IUT after it is tested, and we use this notion to find optimal test suites. Finally, in Section 6 we present our conclusions and some lines of future work.

2 Basic notions

In this section we present some basic notions used in the paper. First, we introduce some statistics notions. An *event* is any reaction we can detect from a system or environment; a random variable is a function associating each event with its probability.

Definition 1. Let $\mathcal{A}$ be a set of events and $\xi : \mathcal{A} \to [0, 1]$ be a function such that $\sum_{\alpha \in \mathcal{A}} \xi(\alpha) = 1$. We say that ξ is a *random variable* for the set of events $\mathcal{A}$.

If we observe that the event $\alpha \in \mathcal{A}$ is produced by a random source whose probabilistic behavior is given by ξ then we say that α *has been generated by* ξ. We extend this notion to sequences of events as expected: If we observe that the sequence of events $H = \langle \alpha_1, \ldots, \alpha_n \rangle$ is consecutively produced by a random source whose probabilistic behavior is given by ξ then we say that H *has been generated by* ξ or that H is a *sample* of ξ.

Given the random variable ξ and a sequence of events H, we denote the *confidence* that H is *generated by* ξ by $\gamma(\xi, H)$. □

This definition introduces a simple version of discrete random variable where all the events are independent. The actual definition of a *random variable* is more complex but it is pointless to use its generality in our setting. In the previous definition, the application of a suitable *hypothesis contrast* is abstracted by the function γ. We have that $\gamma(\xi, H)$ takes a value in $[0, 1]$. Intuitively, a sample will be *rejected* if the probability of observing that sample from a given random variable is low. In the appendix of this paper we present a working definition of the function γ. It is worth to point out that the results of this paper do not depend on the formulation of γ, being possible to *abstract* the actual definition.

Next we present the formalism we will use to define *specifications* and *implementations*. A *probabilistic finite state machine* is a finite state machine where each transition is equipped with a probability denoting its probabilistic propensity. Thus, a transition $s \xrightarrow{i/o}_p s'$ denotes that, if the machine is in state s and the input i is received then, with probability p, it moves to the state s' and produces the output o. We will assume that the environment stimulates the machine with a single input at any time.

Given an input, the machine probabilistically chooses the transition it takes from its current state. Hence, the probability of a transition allows to compare its propensity with the one of any other transition departing from the same state and receiving the *same* input. That is, given s and i, the addition of all values p such that there exist o, s' with $s \xrightarrow{i/o}_p s'$ must be equal to 1. In contrast, there is no requirement binding the probabilities departing from the same state and receiving different inputs because each one describes (part of) a different probabilistic choice of the machine. Thus, we consider a *reactive* interpretation of probabilities (see [7, 16]).

Definition 2. A *Probabilistic Finite State Machine*, in short `PFSM`, is a tuple $M = (S, I, O, \delta, s_0)$ where

- S is the finite *set of states* and $s_0 \in S$ is the *initial state*.
- I and O, with $I \cap O = \varnothing$, denote the sets of *input* and *output* actions, respectively.
- $\delta \subseteq S \times I \times O \times (0, 1] \times S$ is the *set of transitions*. We will write $s \xrightarrow{i/o}_p s'$ to denote $(s, i, o, p, s') \in \delta$.

Transitions and states fulfill the following additional conditions:

- For all $s \in S$ and $i \in I$, the probabilities associated with outgoing transitions add up to 1, that is, $\sum\{p \mid \exists\, o \in O,\, s' \in S :\ s \xrightarrow{i/o}_p s'\} = 1$.
- `PFSMs` are *free of non-observable non-determinism*, that is, if we have two transitions $s \xrightarrow{i/o}_{p_1} s_1$ and $s \xrightarrow{i/o}_{p_2} s_2$ then $p_1 = p_2$ and $s_1 = s_2$.
- In addition, we will assume that implementations are *input-enabled*, that is, for all state s and input i there exist o, p, s' such that $s \xrightarrow{i/o}_p s'$. □

Although `PFSMs` will be used to define specifications and implementations, a different formalism will be used to define *user models*. Specifically, we will use *probabilistic labeled transition systems*. A user model represents the external environment of a system. User models actively produce inputs that stimulate the system, while passively receive outputs produced by the system as a response. The states of a user model are split into two categories: *Input states* and *output states*. In input states, all outgoing transitions denote a different input action. Since inputs are probabilistically chosen by user models, any input transition is endowed with a probability. In particular, $s \xrightarrow{i}_p s'$ denotes that, with probability p, in the input state s, the input i is produced and the state is moved to s'. Given an input state s, the addition of all probabilities p such that there exists i, s' with $s \xrightarrow{i}_p s'$ must be *lower* than or equal to 1. If it is lower then we will consider that the remainder up to 1 implicitly denotes the probability that the interaction with the system *finishes* at the current state. Regarding output states, all transitions departing from an output state are labeled by a different output action. However, output transitions do not have any probability value (let us remind that outputs are chosen by the system). Input and output states will strictly alternate, that is, for any input state s, with $s \xrightarrow{i}_p s'$, s' is an output state, and for any output state s, with $s \xrightarrow{o} s'$, s' is an input state.

Definition 3. A *probabilistic labeled transition system*, in short `PLTS`, is a tuple $U = (S_I, S_O, I, O, \delta, s_0)$ where

- S_I and S_O, with $S_I \cap S_O = \varnothing$, are the finite sets of *input* and *output* states, respectively. $s_0 \in S_I$ is the *initial state*.

- I and O, with $I \cap O = \varnothing$, are the sets of *input* and *output* actions, respectively.
- $\delta \subseteq (S_I \times I \times (0,1] \times S_O) \cup (S_O \times O \times S_I)$ is the *transition relation*. We will write $s \xrightarrow{i}_p s'$ to denote $(s, i, p, s') \in S_I \times I \times (0,1] \times S_O$ and $s \xrightarrow{o} s'$ to denote $(s, o, s') \in S_O \times O \times S_I$.

Transitions and states fulfill the following additional conditions:

- For all input states $s \in S_I$ and input actions $i \in I$ there exists at most one outgoing transition from s: $|\{s \xrightarrow{i}_p s' \mid \exists\, p \in (0,1],\, s' \in S_O\}| \leq 1$.
- For all output states $s \in S_O$ and output actions $o \in O$ there exists exactly one outgoing transition labeled with o: $|\{s \xrightarrow{o} s' \mid \exists\, s' \in S_I\}| = 1$.
- For all input state $s \in S_I$ the addition of the probabilities associated with the outgoing transitions is lower than or equal to 1, that is, $\texttt{cont}(s) = \sum\{p \mid \exists\, s' \in S_O :\ s \xrightarrow{i}_p s'\} \leq 1$. So, the probability of stopping at that state s is $\texttt{stop}(s) = 1 - \texttt{cont}(s)$. □

By iteratively executing transitions, both `PFSMs` and `PLTSs` can produce sequences of inputs and outputs. The probabilities of these sequences are given by the probabilities of the transitions. Next we introduce some *trace* notions. A *probability trace* is a sequence of probabilities, a *trace* is a sequence of inputs and outputs, and a *probabilistic trace* is a tuple containing both.

Definition 4. A *probability trace* π is a finite sequence of probabilities, that is, a possibly empty sequence $\langle p_1, p_2, \ldots, p_n \rangle \in (0,1]^*$. The symbol ϵ denotes the empty probability trace. Let $\pi = \langle p_1, p_2, \ldots, p_n \rangle$ be a probability trace. We define its *sef-product*, denoted by $\prod \pi$, as $\prod_{1 \leq i \leq n} p_i$. Since $\prod_{a \in \varnothing} = 1$, we have $\prod \epsilon = 1$. Let $\pi = \langle p_1, p_2, \ldots, p_n \rangle$ and $\pi' = \langle p'_1, p'_2, \ldots, p'_m \rangle$ be probability traces. Then, $\pi \cdot \pi'$ denotes their concatenation that is, $\langle p_1, p_2, \ldots, p_n, p'_1, p'_2, \ldots, p'_m \rangle$, while $\pi * \pi'$ and π / π' denote their pairwise product and division respectively, that is, $\langle p_1 * p'_1, p_2 * p'_2, \ldots, p_r * p'_r \rangle$ and $\langle p_1 / p'_1, p_2 / p'_2, \ldots, p_r / p'_r \rangle$, where $r = \min(n, m)$.

A *trace* ρ is a finite sequence of input/output actions $(i_1/o_1, i_2/o_2, \ldots, i_n/o_n)$. The symbol ϵ denotes the empty trace. Let ρ and ρ' be traces. Then, $\rho \cdot \rho'$ denotes their concatenation. A *probabilistic trace* is a pair (ρ, π) where ρ is a trace $(i_1/o_1, i_2/o_2, \ldots, i_n/o_n)$ and $\pi = \langle p_1, p_2, \ldots, p_n \rangle$ is a probability trace. If ρ and π are both empty then we have the *empty probabilistic trace*, written as (ϵ, ϵ). Let (ρ, π) and (ρ', π') be probabilistic traces. Then, $(\rho, \pi) \cdot (\rho', \pi')$ denotes their concatenation, that is, $(\rho \cdot \rho', \pi \cdot \pi')$.

An *input trace* ϱ is a finite sequence of input actions $(i_1, i_2, \ldots, i_n)$. We extend the previous notions of empty trace and concatenations to input traces in the expected way. If $\rho = (i_1/o_1, i_2/o_2, \ldots, i_n/o_n)$ then we denote by $\texttt{i}(\rho)$ the input trace $(i_1, i_2, \ldots, i_n)$. A *probabilistic input trace* is a pair (ϱ, π) where ϱ is an input trace $(i_1, i_2, \ldots, i_n)$ and $\pi = \langle p_1, p_2, \ldots, p_n \rangle$. We also consider the concepts of concatenation and empty probabilistic input traces. □

Next we define how to extract traces from `PFSMs` and `PLTSs`. First, we consider the reflexive and transitive closure of the transition relation, and we call it *generalized* transition. Then, probabilistic traces are constructed from generalized transitions by considering their sequences of actions and probabilities.

Definition 5. Let $M = (S, I, O, \delta, s_0)$ be a `PFSM`. We inductively define the *generalized* transitions of M as follows:

– We have that $s \stackrel{\epsilon}{\Longrightarrow}_{\epsilon} s$ is a generalized transition of M for all $s \in S$.
– If $s \stackrel{\rho}{\Longrightarrow}_{\pi} s'$ and $s' \xrightarrow{i/o}_{p} s_1$ then $s \stackrel{\rho \cdot i/o}{\Longrightarrow}_{\pi \cdot \langle p \rangle} s_1$ is a generalized transition of M.

We say that (ρ, π) is a *probabilistic trace* of M if there exists $s \in S$ such that $s_0 \stackrel{\rho}{\Longrightarrow}_{\pi} s$. In addition, we say that ρ is a *trace* of M and that $\mathtt{i}(\rho)$ is an *input trace* of M. The sets $\mathtt{pTr}(M)$, $\mathtt{tr}(M)$, $\mathtt{iTr}(M)$ denote the sets of *probabilistic traces*, *traces*, and *input traces* of M, respectively. □

The previous notions can also be defined for PLTSs. In order to obtain sequences of paired inputs and outputs, traces begin and end at input states. Generalized transitions are constructed by taking *pairs* of consecutive PLTS transitions.

Definition 6. Let $U = (S_I, S_O, I, O, \delta, s_0)$ be a PLTS. We inductively define the *generalized* transitions of U as follows:

– We have that $s \stackrel{\epsilon}{\Longrightarrow}_{\epsilon} s$ is a generalized transition of U for all $s \in S_I$.
– If $s \in S_I$, $s \stackrel{\rho}{\Longrightarrow}_{\pi} s'$, and $s' \xrightarrow{i}_{p} s'' \xrightarrow{o} s_1$ then $s \stackrel{\rho \cdot i/o}{\Longrightarrow}_{\pi \cdot \langle p \rangle} s_1$ is a generalized transition of U.

We say that (ρ, π) is a *probabilistic trace* of U if there exists $s \in S_I$ such that $s_0 \stackrel{\rho}{\Longrightarrow}_{\pi} s$. In that case we will also say that $(\mathtt{i}(\rho), \pi)$ is a *probabilistic input trace* of U. In addition, we say that ρ is a *trace* of U and that $\mathtt{i}(\rho)$ is an *input trace* of U. We define the probability of U to stop after ρ, denoted by $\mathtt{stop}_U(\rho)$, as $\mathtt{stop}(s)$. The sets $\mathtt{pTr}(U)$, $\mathtt{piTr}(U)$, $\mathtt{tr}(U)$ and $\mathtt{iTr}(U)$ denote the set of *probabilistic traces*, *traces*, and *input traces* of U respectively. □

Next we identify PLTS that *terminate*, that is, such that all infinite traces have probability 0.

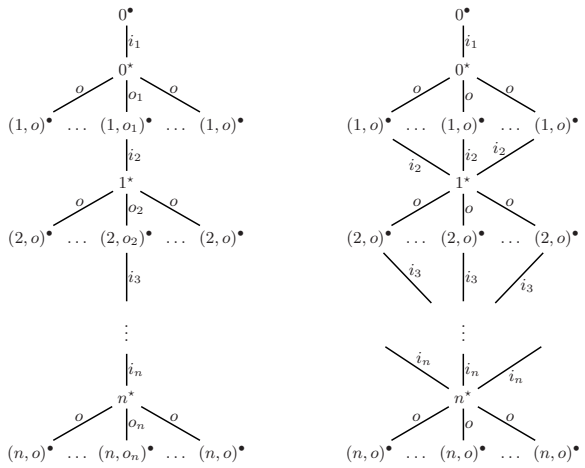
Definition 7. Let U be a PLTS. We say that U is a *terminating* PLTS if for all s such that there exists ρ and π with $s_0 \stackrel{\rho}{\Longrightarrow}_{\pi} s$ we have that there exists s', ρ', π' such that $s \stackrel{\rho'}{\Longrightarrow}_{\pi'} s'$ and $\mathtt{stop}_U(\rho \cdot \rho') > 0$. □

Proposition 1. A PLTS U is terminating iff $\sum_{(\rho,\pi) \in \mathtt{pTr}(U)} (\prod \pi) * \mathtt{stop}_U(\rho) = 1$ □

As we will see, PLTS will be used to denote user models. In particular, any user model will be supposed to be a *terminating* PLTS.

3 Tests and composition of machines

In this section we define our tests as well as the interaction between the notions introduced in the previous section (PFSMs and PLTSs). As we said before, we will use PLTSs to define the behavior of the external environment of a system, that is, a user model. Moreover, PLTSs are also appropriate to define the *tests* we will apply to an IUT. Tests are PLTSs fulfilling some additional conditions. Basically, a test defines a finite sequence of inputs; we will use them to check a given secuence of inputs. Since tests consider a single sequence of inputs, each intermediate input state of the sequence contains a single outgoing transition labeled by the next input and probability 1. Output states offer transitions with different outputs.



a.) $\rho = (i_1/o_1, i_2/o_2, \ldots, i_n/o_n)$ b.) $\varrho = (i_1, i_2, \ldots, i_n)$

Fig. 1. Tests: a.) the ones in [12], and b.) the new ones.

Definition 8. A test $T = (S_I, S_O, I, O, \delta, s_0)$ is a PLTS such that for all $s \in S_I$ there is at most one transition $s \xrightarrow{i}_p s'$ (and if it exists then $p = 1$), and for all $s \in S_O$ there is at most one *next input state* $s \xrightarrow{o} s'$ *with a continuation*, that is, $|\{s'' \mid \exists\, i \in I,\, o \in O,\, s''' \in S_O,\, p \in (0,1] :\ s \xrightarrow{o} s'' \xrightarrow{i}_p s'''\}| \leq 1$. □

Let us note that, contrarily to other frameworks, tests are not provided with diagnostic capabilities on *their own*. In other words, tests do not have fail/success states. Since our framework is probabilistic, the requirements defined by specifications are given in probabilistic terms. Moreover, the absence of transitions labeled by specific outputs in specification states is considered in probabilistic terms as well: If there exists a state s, an input i, and an output o such that there do not exist p, s' with $s \xrightarrow{i/o}_p s'$ then we will consider that the probability of producing o in the state s after receiving the input i is 0. As we will see in the next section, deciding whether the IUT conforms to the specification will also be done in probabilistic terms. In particular, we will consider whether it is *feasible* that the IUT *behaves* as if it were defined as the specification indicates. We will check this fact by means of a suitable *hypothesis contrast*.

Our testing methodology consists in testing the behavior of a system under the assumption that it is stimulated by a given user model. Thus, tests will be extracted from the behavior of the user model. Next we show how a test is constructed from a probabilistic trace of a user model. The input and output states of the test are identified with natural numbers. All the input states (but the first one) are also endowed with an output action. In order to distinguish between input and output states we decorate them with $^\bullet$ and *, respectively. Tests extracted from user model sequences fulfill an additional condition: All input states reached from a given output state (via different outputs) are connected with the *same* output state through the same input, up to the end of the sequence. This approach differs from the tests derived in [12] as well as

Fig. 2. Normalization if composition of PFSMs and PLTSs.

in other methodologies, where all input states *but one* terminate the interaction and only one input state proposes the following input. The form of tests changes in this paper because we decompose the process of testing an IUT into independent tasks, consisting each task in testing the responses of the IUT to a given sequence of inputs. The new form of tests allows to carry out these tasks in a modular way: A single test can process *any* answer to a given sequence of inputs, that is, it detects any sequence of outputs produced by the IUT as response (see the Figure 1.a). On the other hand (Figure 1.b), the tests commented from [12] provide a complete response only if each input i_k is followed by the corresponding output o_k indicated in the trace ρ.

Definition 9. Let $\varrho = (i_1, i_2, \ldots, i_r)$ be an input trace, I be a set of input actions such that $\{i_1, \ldots i_r\} \subseteq I$, and O be a set of output actions. We define the *test associated* to ϱ, denoted by $\texttt{assoc}(\varrho)$, as the test $(S_{IT}, S_{OT}, I, O, \delta_T, 0^\bullet)$, where

- $S_{IT} = \{0^\bullet, r^\bullet\} \cup \{(j,o)^\bullet | o \in O, 1 \leq j \leq r\}$ and $S_{OT} = \{j^\star | 1 \leq j \leq r\}$.
- For all $1 \leq j < r, o \in O$: $(j,o)^\bullet \xrightarrow{i_{j+1}}_1 (j+1)^\star$, $j^\star \xrightarrow{o} (j,o)^\bullet \in \delta_T$. We also have $0^\bullet \xrightarrow{i_1}_1 0^\star$. □

Next we define the composition of a PFSM (denoting either a specification or an IUT) with a PLTS (denoting either a user model or a test) in terms of its behavior, that is, in terms of traces and probabilistic traces. The set of traces is easily computed as the intersection of the traces produced by both components. In order to define the set of probabilistic traces, the ones provided by both components are considered. For a given input/output pair i/o, the probability of producing i will be taken from the corresponding transition of the PLTS, while the probability of producing o as a response to i will be given by a transition of the PFSM. Let us note that the states of a specification do not necessarily define outgoing transitions for all available inputs, that is, specifications are not necessarily *input-enabled*. So, a PFSM representing a specification could not provide a response for an input produced by a PLTS. Since the specification does not define any behavior in this case, we will assume that the PFSM is allowed to produce *any* behavior from this point on. The composition of a PLTS and a PFSM will be constructed to check whether the traces *defined* by the specification are correctly produced by the implementation (under the assumption that these machines are stimulated by the user model). Hence, undefined behaviors will not be considered relevant and will not provide any trace to the composition of the PLTS and the PFSM. In order to appropriately represent the probabilities of the relevant traces, their probabilities will be *normalized* if undefined behaviors appear. We illustrate this process in the following example.

Example 1. Let us suppose that a user model can produce the inputs i_1, i_2, and i_3 with probabilities $\frac{1}{2}$, $\frac{1}{4}$ and $\frac{1}{4}$, respectively (see Figure 2, left). At the same time, the

corresponding specification provides outgoing transitions with inputs i_1 and i_2, but not with i_3 (see Figure 2, right). Since the specification does not define any reaction to i_3, the probabilities of taking inputs i_1 or i_2 in the composition of the specification and the user model are normalized to denote that i_3 is not considered. So, the probability of i_1 becomes $\frac{1/2}{3/4} = \frac{2}{3}$ while the probability of i_2 is $\frac{1/4}{3/4} = \frac{1}{3}$. □

The next definition finds an appropriate normalization factor when these situations appear (in the previous example, this factor is $\frac{3}{4}$). Besides, we show how to recompute the probabilities of all traces in a PLTS when only sequences of inputs that are accepted by a a given PLTS are considered. Finally, we consider the behavior of the composition of a PFSM and a PLTS. The set of traces of this composition is provided by the intersection of the set of traces of each machine. In order find the probabilistic traces we consider, on the one hand, the probabilistic traces of the PFSM and, on the other hand, the probabilistic traces of the PLTS *normalized to* this PFSM.

Definition 10. Let $M = (S_M, I, O, \delta_M, s_{0M})$ be a PFSM and let us consider a PLTS $U = (S_{IU}, S_{OU}, I, O, \delta_U, s_{0U})$ such that $s_{0M} \stackrel{\rho}{\Longrightarrow}_{\pi_1} s_1$ and $s_{0U} \stackrel{\rho}{\Longrightarrow}_{\pi_2} s_2$. We define:

- The sum of the probabilities of *continuing together after* ρ as

$$\mathtt{cont}_{M\|U}(\rho) = \sum \left\{ p \;\middle|\; \begin{array}{c} \exists i \in I,\, o \in O,\, s_2' \in S_{OU},\, s_1' \in S_M,\, r \in (0,1] : \\ s_2 \xrightarrow{i}_p s_2' \;\; \wedge \;\; s_1 \xrightarrow{i/o}_r s_1' \end{array} \right\}$$

- The *normalization factor of* $M \,\|\, U$ *after* ρ as the sum of the previous probability plus the probability of U to stop after ρ, that is $\mathtt{norm}_{M\|U}(\rho) = \mathtt{cont}_{M\|U}(\rho) + \mathtt{stop}_U(\rho)$.

We inductively define the probabilistic traces of U *normalized* to M as follows:

- (ϵ, ϵ) is a normalized probabilistic trace.
- Let (ρ, π) be a normalized probabilistic trace. Let us suppose that we have $s_{0M} \stackrel{\rho}{\Longrightarrow}_{\pi_1} s_1' \xrightarrow{i/o}_{p_1} s_1$ and $s_{0U} \stackrel{\rho}{\Longrightarrow}_{\pi_2} s_2' \; s_2' \xrightarrow{i}_{p_2} s'' \xrightarrow{o} s_2$. Then, $(\rho \cdot i/o, \pi \cdot \langle p \rangle)$ is a normalized probabilistic trace, where p is the product of p_1 and p_2 *normalized* with respect to the normalization factor of $M \,\|\, U$ after ρ, that is, $p = \frac{p_1 \cdot p_2}{\mathtt{norm}_{M\|U}(\rho)}$.

Let (ρ, π) be a normalized probabilistic trace where we have $s_{0U} \stackrel{\rho}{\Longrightarrow}_{\pi'} s$ for some π', s. We say that $(\mathtt{i}(\rho), \pi)$ is a *normalized probabilistic input trace*. In addition, we say that ρ is a *normalized trace* and that $\mathtt{i}(\rho)$ is a *normalized input trace*. We define the probability of U to stop after ρ *normalized* to M, denoted by $\mathtt{nstop}_{U,M}(\rho)$, as $\frac{\mathtt{stop}(s)}{\mathtt{norm}_{M\|U}(\rho)}$. The sets $\mathtt{npTr}_M(U)$, $\mathtt{npiTr}(U, M)$, $\mathtt{ntr}(U, M)$ and $\mathtt{niTr}(U, M)$ denote the set of *normalized probabilistic traces*, *normalized traces*, and *normalized input traces* of U to M respectively.

The *set of traces* generated by the *composition* of M and U, denoted by $\mathtt{tr}(M \,\|\, U)$, is defined as $\mathtt{tr}(M) \cap \mathtt{tr}(U)$. The *set of probabilistic traces* generated by the *composition* of M and U, denoted by $\mathtt{pTr}(M \,\|\, U)$, is defined as

$$\{(\rho, \pi_1 * \pi_2) | (\rho, \pi_1) \in \mathtt{pTr}(M) \;\wedge\; (\rho, \pi_2) \in \mathtt{npTr}_M(U)\}$$

The *set of input traces* generated by the *composition* of M and U, denoted by $\mathtt{iTr}(M \,\|\, U)$, is defined as the set $\{\mathtt{i}(\rho) \mid \rho \in \mathtt{tr}(M \,\|\, U)\}$. □

Proposition 2. Let M be a PFSM and let U be a PLTS, then

$$\mathtt{tr}(M \parallel U) = \{\rho \mid \exists p \in (0,1] : \ (\rho, p) \in \mathtt{pTr}(M \parallel U)\}$$ □

Let us remark that the probabilistic behavior of the traces belonging to the composition of PFSMs and PLTSs is completely specified: The probabilities of inputs are provided by the PLTS while the probabilities of outputs are given by the PFSM.

Since our method consists in testing the behavior of the IUT for some sequences of inputs, we will be interested in taking those traces that share a given sequence of inputs. Next we develop these ideas for sequences and sets of sequences.

Definition 11. Let Tr be a set of traces and ϱ an input trace. We define the *set of traces of* Tr *modulo* ϱ, denoted by $\mathtt{tr}_\varrho(Tr)$, as the set $\{\rho \mid \mathtt{i}(\rho) = \varrho, \ \rho \in Tr\}$. If M is a PFSM and U is a PLTS, for the sake of clarity, we write $\mathtt{tr}_\varrho(M)$, $\mathtt{tr}_\varrho(U)$, and $\mathtt{tr}_\varrho(M \parallel U)$ instead of $\mathtt{tr}_\varrho(\mathtt{tr}(M))$, $\mathtt{tr}_\varrho(\mathtt{tr}(U))$, and $\mathtt{tr}_\varrho(\mathtt{tr}(M \parallel U))$, respectively.

Let Tr be a set of traces and iTr a set of input traces. We define the *set of traces of* Tr *modulo* iTr, denoted by $\mathtt{prob}_{pTr}(iTr)$, as the set $\{\mathtt{tr}_\varrho(Tr) | \varrho \in iTr\}$. □

We will construct a random variable denoting the probability of each trace in the composition of a specification and a user. Unfortunately, taking the probability associated to each trace in the composition is not appropriate. In fact, the sum of the probabilities of all traces may be higher than 1. This is because traces denote events such that some of them *include* others. For instance, if the event $(a/b, c/d)$ is produced then we know that (a/b) is also produced. We solve this problem by taking into account a factor that is not explicitly considered in the traces: The choice of a user to stop in a state. In particular, the event representing that $(a/b, c/d)$ is produced *and*, afterwards immediately, the user finishes does not imply that (a/b) is produced and then the user stops. By explicitly considering the termination of traces, we will obtain events such that addition of their probabilities is 1.

Proposition 3. Let M be a PFSM and let U be a terminating PLTS. We have

$$\sum_{(\rho,\pi) \in \mathtt{pTr}(M \| U)} \left(\prod \pi\right) * \mathtt{nstop}_{U,M}(\rho) = 1$$ □

By the previous result, we can use traces *up to termination* to construct a random variable denoting the probability of observe any trace in the composition of a specification and a user.

Definition 12. Let M be a PFSM and let U be PLTS. We define the *traces random variable of the composition of* M *and* U as the function $\xi_{M \| U} : \mathtt{pTr}(M \parallel U) \longrightarrow (0,1]$ such that for all $(\rho, \pi) \in \mathtt{pTr}(M \parallel U)$ we have

$$\xi_{M \| U}(\rho) = \left(\prod \pi\right) * \mathtt{nstop}_{U,M}(\rho)$$ □

If M denotes a specification and U a user model then the previous random variable provides a source to *randomly* generate tests. As we showed before, a test is constructed by following a specific sequence of inputs of the user model; this test allows to detect any sequence of outputs the IUT can produce as response. The random selection of tests can be represented by a random variable associating each test with the probability that some traces are taken in the composition of the specification and the user model. These traces are those such that the sequence of inputs coincides with the sequence of inputs considered by the test.

4 Probabilistic relations

In this section we introduce our probabilistic conformance relations. Following our user customized approach, they relate an IUT *and* a user model with a specification *and* the same user model. These three elements will be related if the probabilistic behavior shown by the IUT when stimulated by the user model appropriately follows the corresponding behavior of the specification. In particular, we will compare the *probabilistic traces* of the composition of the IUT and the user with those corresponding to the composition of the specification and the user. Let us remind that IUTs are input-enabled but specifications might not be so. So, the IUT could define probabilistic traces including sequences of inputs that are not defined in the specification. Since there are no specification requirements for them, these behaviors will be ignored by the relation. In order to do it, an appropriate subset of the traces of the composition of the IUT and the user must be taken. The probability of each trace belonging to this set will be recomputed by considering a suitable normalization. In the following relation, we require that the probabilities of the corresponding traces are *exactly* the same in both compositions. Later we will see another relation where, due to practical reasons, this requirement will be relaxed.

Definition 13. Let S, I be PFSMs and U be a PLTS. We define the *set of probabilistic traces generated by the implementation I and the user model U modulo the specification S*, denoted by $\mathtt{pTr}(I \parallel U)_S$ as the set

$$\{(\rho, \pi_i * \pi_o) \mid \mathtt{i}(\rho) \in \mathtt{iTr}(S) \ \wedge \ (\rho, \pi_i) \in \mathtt{npTr}_S(U) \ \wedge \ (\rho, \pi_o) \in \mathtt{pTr}(I)\}$$

Let S, I be PFSMs and U be a PLTS. We say that I *conforms to S with respect to U*, denoted by $I \ \mathtt{conf}_U \ S$, if $\mathtt{pTr}(I \parallel U)_S = \mathtt{pTr}(S \parallel U)$. □

The previous result provides a diagnostic by comparing the complete set of traces of the composition of the specification and the user with the full set of traces of the implementation and the user (up to the specification). We can also perform local comparisons: A local diagnostic is obtained by comparing only those traces that have a given sequence of inputs. Though we can compare these traces by comparing their corresponding probabilities, we will manipulate these probabilities before. In particular, we will divide the probability of each of them by the probability its sequence of inputs. These values will denote the probability of performing the sequence of outputs of the trace *provided that* the sequence of inputs is the considered one. Though this transformation is not needed to perform the current comparison, using these probabilities will be useful in further analyses.

Definition 14. Let A be a set of probabilistic traces and (ϱ, π) be a probabilistic input trace. We define *the restriction A to (ϱ, π)*, denoted by $A \backslash (\varrho, \pi)$, as the set $\{(\rho, \pi'/\pi) \,|\, (\rho, \pi') \in A \ \wedge \ \mathtt{i}(\rho) = \varrho\}$. □

Definition 15. Let S, I be PFSMs, U be a PLTS, and $(\varrho, \pi) \in \mathtt{npiTr}(U, S)$ such that $\varrho \in \mathtt{iTr}(S)$. We say that I *conforms to S with respect to U in the input trace ϱ*, denoted by $I \ \mathtt{conf}_{U,\varrho} \ S$, if $\mathtt{pTr}(I \parallel U)_S \backslash (\varrho, \pi) = \mathtt{pTr}(S \parallel U) \backslash (\varrho, \pi)$. □

Next we relate our notions of conformance and conformance for a given sequence of inputs. If we have local conformance for all sequences of inputs, then the global conformance is met.

Proposition 4. Let S, I be PFSMs, and U and be a PLTS, then $I\ \mathtt{conf}_U\ S$ iff for any probabilistic input trace $(\varrho, \pi) \in \mathtt{piTr}(U)$ such that $\varrho \in \mathtt{iTr}(S)$ we have $I\ \mathtt{conf}_{U,\varrho}\ S$. □

Our tests are designed to check any input trace. The parallel composition of the test with the specification $S \parallel T$ performs traces that are not present in the parallel composition of the user and the specification $S \parallel U$. However, if we remove the probabilities associated to the input trace in the user model then the probability of the traces that are in both compositions is the same. Thus, if the implementation conforms the specification with respect to the test T (i.e. $I\ \mathtt{conf}_T\ S$), then it also conforms the specification with respect to the user in the trace (i.e. $I\ \mathtt{conf}_{U,\varrho}\ S$).

Proposition 5. Let S be a PFSM, U and be a PLTS, $(\varrho, \pi) \in \mathtt{npiTr}(U, S)$ such that $\varrho \in \mathtt{iTr}(S)$, and $T = \mathtt{assoc}(\varrho)$. Then

- For all $\rho \in \mathtt{tr}(S \parallel U) \cap \mathtt{tr}(S \parallel T)$ we have $\xi_{S\parallel T}(\rho) * \prod \pi = \xi_{S\parallel U}(\rho)$
- if $I\ \mathtt{conf}_T\ S$ then $I\ \mathtt{conf}_{U,\varrho}\ S$.

□

Although the previous relation properly defines our probabilistic requirements, it cannot be used in practice because we cannot *read* the probability attached to a transition in a black-box IUT. So, a more applicable version of the relation is required. Let us note that even though a single observation does not provide valuable information about the probability of an IUT trace, an *approximation* to this value can be calculated by interacting a high number of times with the IUT and analyzing its reactions. In particular, we can compare the empirical behavior of the IUT with the ideal behavior defined by the specification and check whether it is *feasible* that the IUT would have behaved like this if, internally, it were defined conforming to the specification. Depending on the empirical observations, this feasibility may be different. The feasibility degree of a set of samples with respect to its ideal probabilistic behavior (defined by a random variable) will be provided by a suitable contrast hypothesis. We will rewrite the previous relation $I\ \mathtt{conf}_T\ S$ in these terms. The new relation will be parameterized by two values: the test that will be used to get the samples and a feasibility threshold.

Definition 16. Let M be a PFSM and U be a PLTS. We say that a sequence $\langle \rho_1, \rho_2, \ldots, \rho_n \rangle$ is a *trace sample* of $M \parallel U$ if it is generated by $\xi_{M\parallel U}$. □

Definition 17. Let S be a PFSM and $H = \langle \rho_1, \rho_2, \ldots, \rho_n \rangle$ be a sequence of traces. H_S denotes the sub-sequence $\langle \rho_{r1}, \rho_{r2}, \ldots, \rho_{rn} \rangle$ of H that contains all the probabilistic traces whose input sequences can be produced by S, that is, $\mathtt{i}(\rho_{ri}) \in \mathtt{iTr}(S)$.

Let S and I be PFSMs, and U be a PLTS. Let $\varrho \in \mathtt{iTr}(S)$, $T = \mathtt{assoc}(\varrho)$, and $H = \langle \rho_1, \rho_2, \ldots, \rho_n \rangle$ be a trace sample of $I \parallel T$, and $0 \leq \alpha \leq 1$. We write $S\ \mathtt{conf}^{\alpha}_{H}\ I$ if $\gamma(\xi_{S\parallel T}, H_S) \geq \alpha$. □

5 Optimal test suites

In this section we will focus on two aspects: How to find a suitable test suite and how to give a metric that allow us to measure the *quality* of test suites. Test suites will we be chosen when they have a *good value* in that metric. So, we first address the problem of choosing a criteria to measure the quality of the test suite.

5.1 Testing quality measurement

Let us suppose that we have a test suite. We apply each test to validate a single trace. Iterating the process for each test we can get a set of *validated* input traces. Since not all the traces are checked, we have to know how accurate is our judgment about the correctness of the specification. We will measure this accuracy in probabilistic terms. We assume that only the tested input traces are correct and that all the others are incorrect. So, the probability of executing one of those untested traces gives us an *upper bound of the probability that the user finds an error in the implementation*. In order to compute this upper bound, we have to calculate the probability with which the user execute one of the tested traces. The complementary of that probability will be the upper bound we are looking for. In order to compute those probabilities we use the random variable $\xi_{S\|U}$. For any tested input trace ϱ, its probability is equal to the probability of the set of traces of the parallel composition $S \parallel U$ whose input traces are those of ϱ.

Definition 18. Let S be a PFSM and U be a PLTS. Then,

1. If $\varrho \in \mathtt{iTr}(S \| U)$ then we denote by $\mathtt{prob}_{S\|U}(\varrho)$ the probability of the set of events $\mathtt{tr}_\varrho(\mathtt{tr}(S \parallel U))$ assigned by the random variable $\xi_{S\|U}$.
2. For any set $iTr \subseteq \mathtt{iTr}(S \parallel U)$, we denote by $\mathtt{prob}_{S\|U}(iTr)$ the probability of the set of events $\mathtt{tr}_{iTr}(\mathtt{pTr}(S \parallel U))$ assigned by the random variable $\xi_{S\|U}$.

□

In the random variable $\xi_{S\|U}$ we consider only full execution of traces, that is, until the user decides to stop. For that reason we have that all events are independent.

Proposition 6. Let S be a PFSM and U be a PLTS. If $\varrho \in \mathtt{iTr}(S \parallel U)$ then

$$\mathtt{prob}_{S\|U}(\varrho) = \sum \left\{ \left(\prod \pi \right) * \mathtt{nstop}_{U,S}(\rho) \mid \mathtt{i}(\rho) = \varrho \ \wedge \ (\rho, \pi) \in \mathtt{pTr}(S \parallel U) \right\}$$

For any set $iTr \subseteq \mathtt{iTr}(S \| U)$ we have $\mathtt{prob}_{S\|U}(iTr) = \sum \left\{ \mathtt{prob}_{S\|U}(\varrho) \mid \varrho \in iTr \right\}$. □

Next we show how to compute the before mentioned upper bound. The scenario is the following. We have validated some input traces by applying tests. From those validated traces iTr, only *safe* traces are considered. The *upper bound that the user finds a error* is calculated by considering that the rest of input traces behaves incorrectly. So, we calculate the probability to execute one of the safe traces, that is $\mathtt{prob}_{S\|U}(iTr)$, being the complementary probability the bound we are looking for. Let us remark that the IUT does not appear in the expression $\mathtt{prob}_{S\|U}(iTr)$. The reason is that we have already tested the implementation in the input traces of the set iTr. Thus, we can assume that the implementation behaves for those traces as indicated by the specification. Besides, we cannot compute that probability from the implementation since it is a *black box*: We can only test it and take samples from it, but we cannot calculate anything else.

Definition 19. Let S, I be PFSMs and U be a PLTS.

- Let ϱ be an input trace, H be a sample of $I \parallel \mathtt{assoc}(\varrho)$, and α be a feasibility degree. We say that ϱ is (H, α)-*tested* if $I \ \mathtt{conf}^{\alpha}_{H} \ S$.
- Let iTr be a set of input traces and $\mathcal{H} = \{H_\varrho \mid \varrho \in iTr,\ H_\varrho \text{ is a sample of } I \parallel \mathtt{assoc}(\varrho)\}$ be a set of samples. We say that iTr $(\mathcal{H}, \alpha)$-*tested* if $I \ \mathtt{conf}^{\alpha}_{H_\varrho} \ S$ for all $\varrho \in iTr$.

Let $iTr \subseteq \mathtt{iTr}(S \parallel U)$ be a $(\mathcal{H}, \alpha)$-*tested* set of input traces for a set of samples $\mathcal{H}$ and a feasibility degree α. Then, the *upper bound of error probability of the user U to find and error in I with respect to the input trace set iTr*, denoted by $\mathtt{ubErr}^{\mathcal{H},\alpha}_{iTr}(I, U)$, is the probabity of executing a trace ρ such that $\mathtt{i}(\rho) \notin iTr$, that is,

$$\mathtt{ubErr}^{\mathcal{H},\alpha}_{iTr}(I, U) = 1 - \mathtt{prob}_{S \parallel U}(iTr)$$ □

5.2 Obtaining a good test suite

Now we give a criteria to choose the *best test suite*. This criteria turns out to be equivalent to the *0/1 knapsack problem*. Due to the intrinsic complexity of that problem, *good enough* test suites will be obtained by applying one of the known suboptimal algorithms.

Since each test checks a single input trace, our test suite will try to minimize the upper bound introduced in Definition 19. So, to find a good test suite is equivalent to find an input trace set iTr that maximizes $\mathtt{prob}_{S \parallel U}(iTr)$. This will be our first criterium to choose our test suite. Obviously, the set that maximizes that probability is the whole set of input traces, that is usually infinite. We need another criteria to limit the number of tests to be applied. It will consist in minimizing the size of tests. Since each tests consists in a sequence of n pairs input/output, it sends and receives exactly n input/output actions. Then, we have that n is the size of the test.

Definition 20. Let $\varrho = (i_1, 1_2, \ldots i_n)$ be an input trace. We say that the *length* of the test $T = \mathtt{assoc}(\varrho)$ is n and we write $\mathtt{length}(T) = n$. Let iTr be a set of input traces. We define the *length* of the set $\mathcal{T} = \{\mathtt{assoc}(\varrho) \mid \varrho \in iTr\}$, denoted by $\mathtt{length}(\mathcal{T})$, as $\sum\{\mathtt{length}(T) \mid T \in \mathcal{T}\}$

Let S be a PFSM and U be a PLTS. Let $n \in \mathbb{N}$ and $iTr \subseteq \mathtt{iTr}(S \parallel U)$. We say that the set of tests $\mathcal{T} = \{\mathtt{assoc}(\varrho) \mid \varrho \in iTr\}$, with $\mathtt{length}(\mathcal{T}) \leq n$, is *$n$-optimum* if there does not exist another set of traces $iTr' \subseteq \mathtt{iTr}(S \parallel U)$ and a set of tests $\mathcal{T}' = \{\mathtt{assoc}(\varrho) \mid \varrho \in iTr'\}$ with $\mathtt{length}(\mathcal{T}') \leq n$ such that $\mathtt{prob}_{S \parallel U}(iTr') > \mathtt{prob}_{S \parallel U}(iTr)$. □

Let us note that, since each trace is independent from the others, the problem to find an n-optimum test suite is equivalent to the 0/1 knapsack problem:

- The total size of the knapsack is n.
- The elements are the input traces $\varrho = (i_1, \ldots, i_r) \in \mathtt{iTr}(S \parallel U)$ such that $r \leq n$.
- The cost of the trace $\varrho = (i_1, \ldots, i_r)$ is r.
- The value of a trace ϱ is $\mathtt{prob}_{S \parallel U}(\varrho)$.

Due to the intrinsic complexity of that problem, it is not feasible to find an n-optimum test suite. However, we can consider one of the suboptimal well-known algorithms to solve the problem (see for example [23]).

5.3 Testing methodology

Finally, let us briefly sketch our testing methodology:

1. We fix n, the combined size of tests belonging to the suite, and α, the feasibility degree to pass the hypotheses contrast.
2. We find a suboptimal test suite $\mathcal{T}$, corresponding to a set of input traces, for the size n.

3. We generate the trace sample H_T for all test $T \in \mathcal{T}$.
4. We consider the set of input traces whose samples pass the hypotheses contrast with the required feasibility degree

$$\mathcal{H} = \{H_T \mid \exists T \in \mathcal{T} : I\,\mathtt{conf}^{\alpha}_{H_T} S\} \quad iTr = \{\varrho \mid \exists T \in \mathcal{T} : H_T \in \mathcal{H} \wedge T = \mathtt{assoc}(\varrho)\}$$

5. We calculate the probability of error $\mathtt{ubErr}^{\mathcal{H},\alpha}_{iTr}(I, U)$.

6 Conclusions and Future Work

In this paper we have presented a formal methodology to test probabilistic systems that are stimulated according to a given user model. In particular, we compare the behavior of a specification when it is stimulated by a user model with the behavior of an IUT when it is stimulated by the same model. By taking into account the probabilities of systems we have that, after a finite test suite is applied to the IUT, we can measure, for a given confidence degree, an upper bound of the probability that a user behaving as the user model finds an error in the IUT. Though a previous work [12] introduces a first approach to compute this metric, this method lies in the idea of comparing a single random variable denoting *all* the behaviors in the composition of the specification and the user with a sample denoting the behavior of the IUT when the user stimulates it. Though this approach is simple, in some situations it could be imprecise: Since traces are extracted from the interaction of the IUT and the user model, a rare probabilistic behavior of the (correct by definition) user model could lead us to reject the IUT. On the contrary, in this paper we develop a new methodology where the behavior of the user model does not affect the samples to be considered. In particular, we separately study the behavior of the IUT for each sequence of inputs. Hence, the frequency of sequences of inputs is not part of the sampled information. This approach requires to use a specific random variable for each sequence of inputs (instead of a single random variable for all traces), as well as separately validating each sample with respect to its corresponding random variable. We use the method to find, for a given number of input actions, a optimal finite test suite, that is, a suite such that if it is passed then the upper bound of error probability is lower that the value obtained with any other test suite of the same size.

As future work, we plan to *compact* the information collected by samples. Let us suppose that a sample $(a/x, b/y, c/z)$ is obtained. This implies that if the sequence of inputs (a, b) would have been offered instead of (a, b, c), then the sequence of outputs (x, y) would have obtained. That is, if the sample $(a/x, b/y, c/z)$ is obtained then the sample $(a/x, b/y)$ is also obtained, as well as (a/x). Hence, by considering all prefixes of a sample, the number of observations for some sequences of inputs increases. Since the precision of hypothesis contrasts is higher when the size of samples is higher, this approach would allow us to improve the precision of our probabilistic method.

References

1. B.S. Bosik and M.U. Uyar. Finite state machine based formal methods in protocol conformance testing. *Computer Networks & ISDN Systems*, 22:7–33, 1991.
2. L. Bottaci and E.S. Mresa. Efficiency of mutation operators and selective mutation strategies: An empirical study. *Software Testing, Verification and Reliability*, 9:205–232, 1999.

3. 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.
4. I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In *CONCUR'90, LNCS 458*, pages 126–140. Springer, 1990.
5. R. Cleaveland, Z. Dayar, S.A. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93–148, 1999.
6. R. de Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
7. 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.
8. R.G. Hamlet. Testing programs with the aid of a compiler. *IEEE Transactions on Software Engineering*, 3:279–290, 1977.
9. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
10. W.E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8:371–379, 1982.
11. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
12. L.F. Llana-Díaz, M. Núñez, and I. Rodríguez. Customized testing for probabilistic systems. In *18th Int. Conf. on Testing Communicating Systems, TestCom 2006, LNCS 3964*, pages 87–102. Springer, 2006.
13. 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.
14. M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.
15. M. Núñez and I. Rodríguez. Encoding `PAMR` into (timed) `EFSMs`. In *22nd IFIP Conf. on Formal Techniques for Networked and Distributed Systems, FORTE 2002, LNCS 2529*, pages 1–16. Springer, 2002.
16. M. Núñez and D. de Frutos. Testing semantics for probabilistic LOTOS. In *Formal Description Techniques VIII*, pages 365–380. Chapman & Hall, 1995.
17. A. Petrenko. Fault model-driven test derivation from finite state models: Annotated bibliography. In *4th Summer School, MOVEP 2000, LNCS 2067*, pages 196–205. Springer, 2001.
18. I. Rodríguez, M.G. Merayo, and M. Núñez. A logic for assessing sets of heterogeneous testing hypotheses. In *18th Int. Conf. on Testing Communicating Systems, TestCom 2006, LNCS 3964*, pages 39–54. Springer, 2006.
19. K. Sayre. Usage model-based automated testing of C++ templates. In *International Conference on Software Engineering. Proceedings of the first international workshop on Advances in model-based testing*, pages 1–5. ACM Press, 2005.
20. R. Segala. Testing probabilistic automata. In *CONCUR'96, LNCS 1119*, pages 299–314. Springer, 1996.
21. M. Stoelinga and F. Vaandrager. A testing scenario for probabilistic automata. In *ICALP 2003, LNCS 2719*, pages 464–477. Springer, 2003.
22. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.
23. V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.
24. G.H. Walton, J.H. Poore, and C.J. Trammell. Statistical testing of software based on a usage model. *Software - Practice & Experience*, 25(1):97–108, 1995.
25. J.A. Whittaker and J.H. Poore. Markov analysis of software specifications. *ACM Transactions on Software Engineering and Methodology*, 2(1):93–106, 1993.

Appendix: Definition of a hypothesis contrast - Pearson's χ^2

Next we show an operative definition of the function γ considered in Definition 1. In particular, it will be based on *Pearson's* χ^2 *contrast*, but other contrasts could be used.

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 χ^2 random variable.
- The number of freedom degrees of χ^2 is $k-1$. In general, this number is equal to $k-r-1$, where 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. In our framework we have $r = 0$ because the model completely specifies the values of p_i before the samples are observed.
- We will *accept* that the sample follows the proposed random variable if the probability to obtain a discrepancy greater 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 $\max\{\alpha \mid X^2 \leq \chi^2_\alpha(k-1)\}$.

Next we define the function γ in terms of the previous mechanism. We will consider two sets of events $\mathcal{A}$ and $\mathcal{A}'$, with $\mathcal{A} \subseteq \mathcal{A}'$. The set $\mathcal{A}$ gives the domain of the random variable ξ, while the events denoted by H belong to $\mathcal{A}'$. If the sample includes any event a that is not considered by the random variable (i.e., $a \notin \mathcal{A}$) then the sample cannot be generated by the random variable and the minimal *feasibility*, that is 0, is returned. Otherwise, we return the maximal feasibility α such that the hypothesis contrast is passed.

Definition 21. Let $\mathcal{A}$ and $\mathcal{A}'$ be sets of events, with $\mathcal{A} \subseteq \mathcal{A}'$, and H be a sample of elements belonging to $\mathcal{A}'$. Let $\xi : \mathcal{A} \to (0, 1]$ be a random variable. We define the confidence of ξ on H, denoted by $\gamma(\xi, H)$, as follows:

$$\gamma(\xi, H) = \begin{cases} 0 & \texttt{if } H \cap \bar{\mathcal{A}} \neq \varnothing \\ \max\{\alpha \mid X^2_\xi \leq \chi^2_\alpha(k-1)\} & \texttt{otherwise} \end{cases}$$

where X^2_ξ denotes the discrepancy level of the sample H on ξ, calculated as explained above by considering that the sampling space is $\mathcal{A}$. □