Mutation Testing from Probabilistic and Stochastic Finite State Machines

Robert M. Hierons [a] Mercedes G. Merayo [b]

[a] *School of Information Systems, and Computing Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH*

[b] *Brunel University and Universidad Complutense de Madrid, 28040 Madrid, Spain*

Abstract

Specification mutation involves mutating a specification, and for each mutation a test is derived that distinguishes the behaviours of the mutated and original specifications. This approach has been applied with finite state machines based models. This paper extends mutation testing to finite state machine models that contain non-functional properties. The paper describes several ways of mutating a finite state machine with probabilities (PFSM) or stochastic time (PSFSM) attached to their transitions and shows how test sequences that distinguish between them and their mutants can be generated. Testing then involves applying each test sequence multiple times, observing the resultant output sequences and using results from statistical sampling theory in order to compare the observed frequency of each output sequence with that expected.

Key words: mutation testing; probabilities; stochastic time; specification mutation

1 Introduction

Traditionally formal methods have concentrated on the representation of the functional behaviour of the systems. Such models are an effective way of representing the required functional properties of the system under test but do not allow us to express desired non-functional properties such as time, probabilities or resources. Several formalisms were extended in order to deal with these kind of properties. The new languages allow the explicit representation of the probability of performing a certain task [34,22,15,11,44] as well as the time consumed by the system while performing tasks, being it either given by fix amounts of time [47,43] or defined in probabilistic/stochastic terms [28,6,37,9].

The models used in this paper include probabilities and stochastic time. Many systems have real-time constraints and thus the inclusion of time is important. Probabilities are highly relevant where resources are shared and so Quality of Service requirements can be probabilistic. In addition, many systems are probabilistic in nature due to either the use of communications over an unreliable medium or through the system consisting of several threads or parallel components and there being different possible synchronization sequences (see, for example, [36]). There are also a number of communications protocols, such as Bluetooth and Ethernet, that have probabilistic requirements [18]. Finally, in order to reason about embedded systems, which are state-based, it is often necessary to use probabilities (see, for example, [42]).

In order to specify systems dealing with probabilities we will use *Probabilistic Finite State Machines (PFSMs)* that are finite state machines with probabilities attached to their transitions. Intuitively, a transition in a finite state machine indicates that if the machine is in a state s and receives an input i then it can produce an output o and 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 $s \xrightarrow{i/o/p} s'$ indicates that the probability with which the event happens is p. We consider a variant of the *reactive* interpretation of probabilities (see for example [34]) since it is the most suitable for our framework. Intuitively, a reactive interpretation imposes a probabilistic relation among transitions labelled by the same action but choices between different actions are not quantified. In our setting we are able to express probabilistic relations between transitions outgoing from a state and having the same input action (the output may vary). For example, let us suppose that the transitions from state s are $t_1 = s \xrightarrow{i_1/o_1/p_1} s_1$, $t_2 = s \xrightarrow{i_1/o_2/p_2} s_2$, $t_3 = s \xrightarrow{i_1/o_3/p_3} s_2$, $t_4 = s \xrightarrow{i_2/o_1/p_4} s_3$, and $t_5 = s \xrightarrow{i_2/o_3/p_5} s_1$. If input i_1 is received then the choice between t_1, t_2, and t_3 will be resolved according to probabilities p_1, p_2, and p_3. Naturally, these values must lie between 0 and 1 and their sum should be 1. Something similar happens for t_4 and t_5. However, there does not exist any probabilistic relation between transitions labelled with different input actions (e.g. t_1 and t_4).

Regarding stochastic models, the main idea is that time information is incremented with some kind of probabilistic information. There are several proposals for timed testing (e.g. [39,14,27,54,19]). In these works time is considered to be *deterministic*. Even though the inclusion of time allows the specifier to give a more precise description of the system to be implemented, there are frequent situations that cannot be accurately described by using fixed amounts of time. For example, we may desire to specify a system where a message is expected to be received with probability $\frac{1}{2}$ in the interval $(0, 1]$, with probability $\frac{1}{4}$ in $(1, 2]$, and so on. Such properties are more general that the usual *deterministic* time where we could only specify that the message arrives at a particular time in the interval $(0, \infty)$. As we have already mentioned, there are several *stochastic* extensions of classical formal models, as process algebras and Petri Nets. However, stochastic testing techniques are very recent. Among the works in this field we note out [5,37,41,4,40], where extensions of the classical theory of testing [17,23] are given.

In order to deal with stochastic time we consider a suitable extension of PFSM, *Probabilistic-Stochastic Finite State Machines (PSFSMs).* In this formalism the time consumed between the input being applied and the output being received is given by *random variables*. That is, instead of having expressions such as "the action o takes t units of time to be performed" we will have expressions such as "with probability p the action o will be performed before t units of time". Thus the interpretation of a transition $s \xrightarrow{i/o/p}_{\xi} s'$ is "if the machine is in state s and receives an input i, then with probability p it will produce the output o and it will change its state to s' before an amount of time t with probability $P(\xi \leq t)$ ".

After describing the formalisms to deal with these concepts we present a *testing methodology* based on mutation testing. Originally *mutation testing* was applied to code [30,8] but some work has looked at *specification mutation* [10]. Here the specification is mutated and for each *mutant* a test is derived that distinguishes the behaviours of the mutated and original specifications. The effect is to ensure that the implementation under test (IUT) does not implement any of the incorrect specifications. Mutations are chosen in order to simulate real faults. The belief is that if a test suite distinguishes between the specification and mutants then it distinguishes between the specification and any faulty IUT. We describe different *mutation operators* that can be applied to a PFSM specification. Additionally, we present approaches to finding input sequences in order to distinguish the mutants and the specification.

This paper concerns black-box testing; if we apply an input to an IUT then we observe an output but we cannot *see* the probabilities that the IUT has assigned to the choices. Thus, even though implementations will behave according to fixed probabilities we cannot determine their values through testing. In our approach, we *estimate* the probabilities by applying a test several times. We use *statistical results* to establish the number of times we need to apply the test to obtaining a required confidence level. Some of the results appearing in this paper have appeared in [25]. The main contribution of this paper with respect to this previous work is to extend the formalisms and results appearing in these papers to deal with stochastic time. It transpires that the problem of deciding whether two PFSMs or PSFSMs are equivalent can be decided in polynomial time. This is important since the equivalent mutant problem causes significant problems in traditional mutation testing but it disappears for the models considered in this paper.

The rest of the paper is organized as follows. In the next section we introduce preliminary concepts and the notion of a PFSM. In Section 3 we show how we can produce input sequences that distinguish states of a PFSM. In Section 4 we introduce mutation operators for PFSMs and corresponding test generation methods. In Section 5 we describe how testing can use input sequences produced by the methods in Section 4. In Section 6 we extend the PFSM in order to deal with stochastic time, introducing the notion of PSFSM. In Section 7 we show how we can produce input sequences that distinguish states of a PSFSM. An algorithm for determining if two PSFA are equivalent is presented. In Section 8 we present a new mutation operator for PSFSM and describe the generation of tests for distinguish the mutant and the original machine. In Section 9 we review previous works on testing probabilistic systems. Finally, in Section 10 we present our conclusions and some lines of future work.

2 Preliminaries

2.1 Basic notation

In this paper sequences are represented by listing their elements preceded by $\langle$, followed by $\rangle$, and separated by commas. Where a variable represents a sequence its name will have a bar above it, an example being $\bar{a}$. In addition, $[0, 1]$ denotes the set $\{p \mid 0 \leq p \leq 1\}$ of numbers that could represent probabilities, $(0, 1]$ denotes the set $\{p \mid 0 < p \leq 1\}$ of numbers that could represent positive probabilities, and $(0, 1)$ denotes the set of values strictly between 0 and 1 and so $(0, 1) = \{p \mid 0 < p < 1\}$.

Definition 1 *Given set X, $\mathcal{P}(X)$ denotes the powerset of X: the set of subsets of X. Thus, $\mathcal{P}(X) = \{X' | X' \subseteq X\}$. Given set W of sequences, $Pre(W) = \{\bar{x}' | \exists \bar{x} \in W, \bar{x}'' \in X^*. \bar{x} = \bar{x}'\bar{x}''\}$ denotes the set of prefixes of sequences from W. Given sets A and B, $A \leftrightarrow B$ denotes the set of relations between A and B. Given a relation f of type $A \leftrightarrow B$ and $a \in A$, a is related to b under f is denoted by $f(a, b)$ and $f(a)$ denotes the set of elements of B related to a under f and so $f(a) = \{b \in B | f(a, b)\}$.*

2.2 Mutation testing

The idea behind mutation testing is that if a test suite distinguishes a program P from other similar programs then it is probably good at discovering faults. The technique introduces small changes in a program, one at a time, to generate a set of *mutants*. We produce mutants by applying one or more *mutation operators* to a given program. In general, P' is an *nth order mutant* if it is produced by applying a sequence of n mutation operators. Usually only *first order mutants* are considered and two arguments are used to justify this. First, the *competent programmer hypothesis* states that expert programmers often write almost correct programs, so low order mutants represent most real faults. Second, if the tests find small differences generated by low order mutants, then it is likely that they find more complex differences. This is called the *coupling effect*.

After we have obtained a collection of mutants from a program, a set of tests T is applied to distinguish each of the mutants from the original program. If the output produced by a mutant P' is different to the one produced by the original program P for test $t \in T$, then t *kills* P'. If no possible test case kills P', then P' is an *equivalent mutant* of P. The objective of mutation analysis is to produce test cases that kill all non-equivalent mutants. Test suites that achieve this goal are *adequate relative to mutation*.

Another strategy given in [10] is *specification mutation*. The specification is mutated, and for each mutation a test is derived that distinguishes the behaviours of the mutated and original specifications. The effect is to ensure that the system under test does not implement any of the incorrect specifications. The mutations are chosen in order to simulate real faults and thus the belief is that a test suite that kills the mutants will not be passed by a faulty system. This approach has also been applied with finite state machines based models [50,49,51,55].

2.3 Finite Automata

A Finite Automaton (FA) N is defined by a tuple (S, s_0, A, δ, S_F) in which S is a finite set of states, $s_0 \in S$ is the initial state, A is the finite alphabet, $\delta : S \times A \leftrightarrow S$ is the state transfer relation, and $S_F \subseteq S$ is the set of final states. If N receives $a \in A$ when in state $s \in S$ it moves to a state $s' \in \delta(s, a)$ and this defines a transition (s, s', a). The relation δ can be extended to take sequences from A^*, giving relation δ^*, in the usual way. FA N is a deterministic finite automaton (DFA) if for all $a \in A$ and $s \in S$, $|\delta(s, a)| \leq 1$.

State s of N defines the language $L_N(s) = \{\bar{a} \in A^* | \delta^*(s, \bar{a}) \cap S_F \neq \emptyset\}$ of words that can take N from s to a final state. Word $\bar{a} \in A^*$ *distinguishes* states s and s' of N if $\bar{a}$ is in exactly one of $L_N(s)$ and $L_N(s')$. If no word distinguishes s and s' then they are *equivalent*. Two FA are *equivalent* if their initial states are equivalent. DFA N is *minimal* if no DFA with fewer states is equivalent to N.

A Probabilistic Finite Automaton (PFA) N is defined by a tuple $(S, s_0, A, \delta, S_F, prob)$ in which S is a finite set of states, $s_0 \in S$ is the initial state, A is the finite alphabet, $\delta : S \times A \leftrightarrow S$ is the state transfer relation, $S_F \subseteq S$ is the set of final states, and $prob$ is the transition probability function of type $S \times A \times S \to [0, 1]$. If N receives $a \in A$ when in state $s \in S$ it moves to a state $s' \in \delta(s, a)$ with probability $prob(s, a, s')$ and this defines transition (s, s', a). The relation δ can be extended to take sequences from A^*, giving δ^*, in the usual way.

Definition 2 *Let $N = (S, s_0, A, \delta, S_F, prob)$ be a PFA and let s, s' be states of N. Then $\bar{a} \in A^*$* distinguishes *states s and s' if the string $\bar{a}$ is accepted from states s and s' with different probabilities. States s and s' are* equivalent *if no string from A^* distinguishes them.*

A non-deterministic finite state machine (NFSM) is a FA in which each transition has an associated output. An NFSM is defined by a tuple (S, s_0, X, Y, f) in which S is a finite set of states, $s_0 \in S$ is the initial state, X is the finite input alphabet, Y is the finite output alphabet, and f is the transition relation. For each state $s \in S$ and input $x \in X$, $f(s, x)$ denotes a set of tuples of the form (s', y) in which $s' \in S$ and $y \in Y$. Given $(s', y) \in f(s, x)$, $(s, s', x/y)$ is a *transition* and this should be interpreted as meaning that if we receive input x while in state s then we can move to state s' and produce output y. A deterministic finite state machine (DFSM) is an NFSM in which for every state s and input x, $|f(s, x)| \leq 1$. There has been much interest in testing from a DFSM (see, for example, [13,26,3]) or an NFSM (see, for example, [38,46,31,24]). See [35] for a survey.

A probabilistic finite state machine (PFSM) is an NFSM in which every transition also has an associated probability.

Definition 3 *A PFSM M is defined by a tuple (S, s_0, X, Y, h) in which S is a finite set of states, $s_0 \in S$ is the initial state, X is the finite input alphabet, Y is the finite output alphabet, and $h : S \times X \leftrightarrow S \times Y \times (0, 1]$ is the transition relation. For each state $s \in S$ and input $x \in X$, $h(s, x)$ denotes a set of tuples of the form (s', y, p) in which $s' \in S$, $y \in Y$, and $p \in (0, 1]$[1]. For all $s \in S$ and $x \in X$, $\sum_{(s',y,p)\in h(s,x)} p = 1$.*

If $(s', y, p) \in h(s, x)$ then (s, x, y, s', p) is a transition of M with starting state s. The probabilities should be interpreted in the following way. If $(s', y, p) \in h(s, x)$ and M receives input x when in state s then with probability p it moves to state s' and produces output y. For a survey on probabilistic automata see [58].

It is worth noting that there cannot exist s, x, y, s' and two different probabilities p_1 and p_2 such that $(s', y, p_1) \in h(s, x)$ and $(s', y, p_2) \in h(s, x)$. Naturally, any such pair of transitions can be represented by means of a unique transition which associated probability $p_1 + p_2$.

Example 1 *Let us consider the probabilistic finite state machine depicted in Figure 1. Each transition has an associated probability. We can observe that all the transitions from state s_2 have probability 1. In contrast, for the state s_1 we have the transitions $(s_1, i_2, o_1, s_2, \frac{1}{4})$ and $(s_1, i_2, o_2, s_3, \frac{3}{4})$ with input i_2; naturally their probabilities sum to 1.*

An alternative characterization of the transitions of M is through a function p_M that provides us with the probability associated with a transition.

[1] An equivalent but less compact representation would include in $h(s, x)$ the transitions with probability 0.

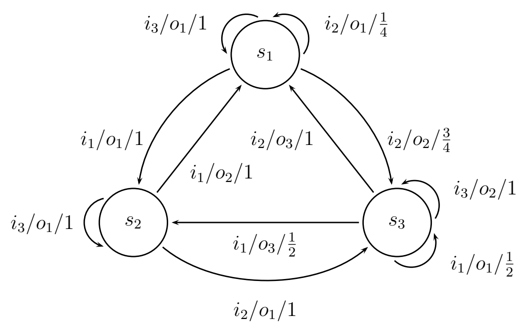


Figure 1. A PFSM

Definition 4 *Let* $M = (S, s_0, X, Y, h)$ *be a PFSM. We define the function* $p_M : S \times X \times Y \times S \to [0, 1]$ *as:*

$$p_M(s, x, y, s') = \begin{cases} p & \text{if } \exists\, p : (s', y, p) \in h(s, x) \\ 0 & \text{otherwise} \end{cases}$$

Given states s and s', input x and output y, $p_M(s, x, y, s')$ is the probability that M moves to state s' and produces output y if it receives input x when in state s. Naturally h and p_M fully define one another since $(s', y, p) \in h(s, x) \Leftrightarrow p > 0 \wedge p_M(s, x, y, s') = p$.

We can extend the transition relation h to input sequences, producing relation h^* of type $S \times X^* \leftrightarrow S \times Y^* \times (0, 1]$. It is simplest to first define p^*_M and express h^* in terms of p^*_M. A similar approach has been applied for PFA (see, for example, [58]).

Definition 5 *Let* $M = (S, s_0, X, Y, h)$ *be a PFSM. Given input/output sequence* $\bar{x}/\bar{y}$ *and state* $s \in S$ *we define the probability of reaching state* s' *from* s *with* $\bar{x}/\bar{y}$ *as:*

$$\begin{aligned} p^*_M(s, \epsilon, y, s') &= \begin{cases} 1 & \textit{if } y = \epsilon \textit{ and } s' = s, \\ 0 & \textit{otherwise} \end{cases} \\ p^*_M(s, \bar{x}x, \bar{y}y, s') &= \sum_{s'' \in S} p^*_M(s, \bar{x}, \bar{y}, s'') p_M(s'', x, y, s') \end{aligned}$$

Let us note that $p^*_M(s, \bar{x}, \bar{y}, s')$ is 0 whenever $|\bar{x}| \neq |\bar{y}|$. In a slight abuse of notation $p^*_M(s, \bar{x}, \bar{y})$ denotes the probability that M produces output sequence $\bar{y}$ if it receives input sequence $\bar{x}$ when in state s. Thus, $p^*_M(s, \bar{x}, \bar{y}) = \sum_{s' \in S} p^*_M(s, \bar{x}, \bar{y}, s')$. We can now extend the transition relation h to input sequences.

$$h^*(s, \bar{x}) = \{(s', \bar{y}, p) | p = p^*_M(s, \bar{x}, \bar{y}, s') \wedge p > 0\}$$

PFSM M is *observable*, or *output-complete*, if for every state s, input x and output y there is at most one transition leaving s with input x and output y. In this paper we consider both observable PFSMs (OPFSMs) and PFSMs that are not observable. PFSM M is *completely specified* if for every state s and input x, $|h(s, x)| \geq 1$ holds.

Some systems have a special operation called a reset that takes the system to its initial state irrespective of the current state. The IUT has a *reliable reset* if it has a reset that is known to be correct. A reliable reset could represent some way of resetting the IUT, such as switching it off and then on again. In this paper we assume that the IUT has a reliable reset.

Two states of an NFSM or DFSM are equivalent if they define the same sets of behaviours: the same set of input/output sequences. For two states of an PFSM to be equivalent we need that they define the same sets of behaviours with the same probabilities. Thus, states s and s' of PFSM M are *equivalent* if for every input sequence $\bar{x} \in X^*$ and output sequence $\bar{y} \in Y^*$, we have that $p^*_M(s, \bar{x}, \bar{y}) = p^*_M(s', \bar{x}, \bar{y})$. If s and s' are not equivalent then they are *distinguishable*. Two PFSMs are equivalent if their initial states are equivalent. If there exists some $\bar{y} \in Y^*$ such that $p^*_M(s, \bar{x}, \bar{y}) \neq p^*_M(s', \bar{x}, \bar{y})$ then $\bar{x}$ is said to *distinguish* s and s'. M is minimal if no PFSM with fewer states that M is equivalent to M.

Let us suppose that we are testing a black box that is equivalent to M and we know that we are either in state s or in state s'. If $\bar{x}$ distinguishes s and s' then it can be used to determine the state if we can apply it in the current state multiple times since we can estimate the probability of each output sequence and the corresponding probabilities are different for s and s'. If we have a reset then we could repeat the following separated by resets: apply the test sequence that led to the current state and then apply $\bar{x}$. If we are to apply an input sequence $\bar{x}$ once only to distinguish two states s and s' then we need a stronger concept: the input of $\bar{x}$ must be guaranteed to lead to different output sequences from s and s' and thus the corresponding sets of possible output sequences must be disjoint.

When reasoning about testing it is normal to assume that the IUT behaves like an unknown element of a fault model (see, for example [32]). Usually the fault model contains descriptions written in the same language as the specification. Thus, for example, when testing from a DFSM it is normal to assume that the IUT behaves like an unknown DFSM. Conformance relations can then be formally defined. Here we briefly review the conformance relations for testing from a (completely specified) DFSM or NFSM.

When testing from a completely specified NFSM there are two notions of correctness. One notion is equivalence, but an alternative is that every behaviour of the IUT is also a behaviour of the specification and that the IUT is completely specified. The assignment of probabilities to transitions removes the possibility of using the second notion of correctness for NFSMs and thus when testing from a PFSM the IUT is correct if it behaves like an unknown PFSM N that is equivalent [2] to M.

3 Distinguishing states of a PFSM

This section shows how we can produce an input sequence that distinguishes two states of PFSM M. We first consider the case where M is observable and show that here the problem can be represented in terms of finding a sequence that distinguishes two states of a FA. We then consider the general case.

Definition 6 *Let $M = (S, s_0, X, Y, h)$ be a PFSM. We define a FA $F(M) = (S, s_0, A, \delta, S)$ where A is $X \times Y \times (0, 1]$ and given $s \in S$, $x \in X$, $y \in Y$, and $p \in (0, 1]$, $\delta(s, (x, y, p)) = s'$ if and only if $(s', y, p) \in h(s, x)$.*

Given $\bar{a} \in A^*$, let $in(\bar{a})$ denote the corresponding input sequence. The function in can be defined recursively in the following way. First, the base case is $in(\epsilon) = \epsilon$. Given $a = (x, y, p) \in A$ and $\bar{a} \in A^*$, $in(a\bar{a}) = xin(\bar{a})$. The following show that algorithms that produce sequences that distinguish states of a FA can also be used to produce sequences that distinguish state of an OPFSM.

Proposition 1 *If input sequence $\bar{x}$ distinguishes states s and s' of OPFSM M then there is some $\bar{a} \in A^*$ such that $\bar{a}$ distinguishes states s and s' of $F(M)$ and $in(\bar{a}) = \bar{x}$.*

Proof:

Let us suppose that $\bar{x}$ distinguishes states s and s' of M. By definition, there exists $\bar{y} \in Y^*$ such that $p^*_M(s, \bar{x}, \bar{y}) \neq p^*_M(s', \bar{x}, \bar{y})$. Without loss of generality, assume that $p = p^*_M(s, \bar{x}, \bar{y}) > 0$.

Since M is observable, there is only one walk in M from state s with input/output sequence $\bar{x}/\bar{y}$. Let $\langle t_1, \ldots, t_k\rangle$ ($k = |\bar{x}|$) denote the sequence of transitions on this walk and for each $t_i = (s_i, x_i, y_i, s_{i+1}, p_i)$ let $a_i = (x_i, y_i, p_i)$ and let $\bar{a} = \langle a_1, \ldots, a_k\rangle$. If there is no walk from s' in M with input/output sequence $\bar{x}/\bar{y}$ then $\bar{a}$ does not label a walk from state s' of $F(M)$ and so $\bar{a}$ distinguishes states s and s' of $F(M)$ as required. Alternatively, if there is a walk from s' in M with input/output sequence $\bar{x}/\bar{y}$ then there is only one such walk so since $p^*_M(s, \bar{x}, \bar{y}) \neq p^*_M(s', \bar{x}, \bar{y})$, $\bar{a}$ does not label a walk from state s' of $F(M)$ and so $\bar{a}$ distinguishes states s and s' of $F(M)$ as required.

[2] A richer set of conformance relations have been defined for general probabilistic state machines (see, for example, [52,59]).

Proposition 2 *Given an OPFSM M and $\bar{a} \in A^*$ such that $\bar{a}$ distinguishes states s and s' of $F(M)$, if no proper prefix of $\bar{a}$ distinguishes s and s' and $in(\bar{a}) = \bar{x}$ then $\bar{x}$ distinguishes states s and s' of M.*

Proof:

Let us suppose that $\bar{a}$ distinguishes states s and s' of $F(M)$ and no proper prefix of $\bar{a}$ distinguishes s and s'. Let $\bar{a} = \langle a_1, \dots, a_k \rangle$, $a_i = (x_i, y_i, p_i)$, $\bar{x} = \langle x_1, \dots x_k \rangle$ and $\bar{y} = \langle y_1, \dots, y_k \rangle$. Without loss of generality, $\bar{a}$ labels a walk from state s of $F(M)$ and does not label a walk from state s' of $F(M)$.

Since M is observable, there is only one walk in M from state s with input/output sequence $\bar{x}/\bar{y}$. If $\bar{x}/\bar{y}$ does not label a walk from state s' of M then by definition $\bar{x}$ distinguishes states s and s' of M as required. We thus assume that $\bar{x}/\bar{y}$ labels a walk from state s' of M and that this walk consists of the sequence $\langle t'_1, \dots, t'_k \rangle$ of transitions. Let p'_i denote the probability associated with transition t'_i. Since no proper prefix of $\bar{a}$ distinguishes s and s' and so $\langle a_1, \dots, a_{k-1} \rangle$ labels a walk from s' in $F(M)$ we have that $p_i = p'_i$ for $1 \leq i < k$. Since $\bar{a}$ does not label a walk from s' in $F(M)$, $p_k \neq p'_k$. Thus, $p^*_M(s, \bar{x}, \bar{y}) = p_1 \dots p_k \neq p'_1 \dots p'_k = p^*_M(s', \bar{x}, \bar{y})$ and so $\bar{x}$ distinguishes states s and s' of M as required.

Note the condition that no proper prefix of $\bar{a}$ distinguishes s and s'. To see why we require this suppose that $\bar{a} = (x_1, y_1, 0.5)(x_2, y_2, 0.5)$, $\bar{a}$ labels a walk from s, and there is a walk with label $(x_1, y_1, 1)(x_2, y_2, 0.25)$ from state s'. Then $\bar{a}$ distinguishes states s and s' of $F(M)$, since $\bar{a}$ labels a walk from s but not s'. However, the input sequence $x_1 x_2$ might not distinguish between states s and s' of M since both states have a probability of 0.25 of producing $y_1 y_2$. Naturally, if we consider the minimal prefix of $\bar{a}$ that distinguishes states s and s' of $F(M)$ then the corresponding input sequence x_1 does distinguish between states s and s' of M.

Proposition 3 *There is an algorithm running in time $O(n^2)$ that takes two states s_1 and s_2 of OPFSM M and determines whether they are equivalent, where n is the number of states of M. If s_1 and s_2 are distinguishable then the algorithm returns a minimal input/output sequence of length no more than $n - 1$ that distinguishes them.*

Proof:

Since $F(M)$ is a minimal FA, there exists a set of sequences of length at most $n-1$ that pairwise distinguish the states of $F(M)$ and such a set can be found in $O(n^2)$ time (see, for example, [21]). The result thus follows from Proposition 2.

More general results have been proved for probabilistic FA that are not deterministic and thus for PFSM that are not observable [3] . The following has been proved [56].

Theorem 1 *There is an algorithm running in time $O((n_1+n_2)^4)$ that takes two probabilistic automata U_1 and U_2 and determines whether U_1 and U_2 are equivalent, where n_1 and n_2 are the number of states of U_1 and U_2 respectively. Furthermore, if U_1 and U_2 are not equivalent then the algorithm outputs the lexicographically minimum string that is accepted by U_1 and U_2 with different probabilities. This string will always be of length at most n_1+n_2-1.*

We now show how this result can be applied to PFSMs.

Definition 7 *Given PFSM $M=(S,s_0,X,Y,h)$ we can define a PFA $F_P(M)=(S_E,s_0,A,\delta,S,prob)$ in which*

(1) $S_E=S\cup\{s_E\}$ where $s_E\notin S$.
(2) The alphabet A is $X\times Y$.
(3) Given $s\in S$, $x\in X$, $y\in Y$, and $p\in(0,1]$, $\delta(s,(x,y))=s'$ and $prob(s,(x,y),s')=p$ if and only if $(s',y,p)\in h(s,x)$.
(4) Given $s\in S$, $x\in X$, and $y\in Y$ we have $\delta(s,(x,y))=s_E$ and $prob(s,(x,y),s_E)=p$ if and only if $p>0$ and $p=1-\sum_{(s'',y,q)\in h(s,x)}q$.

It is worth to note that we need to *adjust* probabilities when we define the associated PFA. In our framework, the probabilities attached to the transitions outgoing from a state and labelled with the same input action sum up to 1. If we only translate the transitions with the associated probabilities and actions to the automaton, it may happen the sum is less than 1 for a label (x,y). Let us consider a machine with three transitions outgoing from a state s: $(s,i_1,o_1,s_1,\frac{1}{4})$, $(s,i_1,o_1,s_2,\frac{1}{2})$, and $(s,i_1,o_2,s_3,\frac{1}{4})$. Then, in the associated automaton we would have that, for example, the probabilities of the transitions labelled with action (i_1,o_1) sum up to $\frac{3}{4}$. We adjust it when we define the associated automaton, including a new state s_E, an *"error state"*, and creating a new transition from each state and action outgoing from it, whose probabilities do not add up to 1. Each such transition has s_E as its final state, has a label with the corresponding action, and has attached a probability equal to the amount required to reach probability 1. In our example, we create three transitions in the automaton outgoing from the state s for the label (i_1,o_1): $(s,i_1,o_1,s_1,\frac{1}{4})$ and $(s,i_1,o_1,s_2,\frac{1}{2})$, corresponding to the ones in the PFSM and an additional transition $(s,i_1,o_1,s_E,\frac{1}{4})$. In this way, the probabilities of the transitions outgoing from state s_1 and labelled with (x,y) sum to 1. Let us point out that the new *error state* is not a final state. The following results are an immediate consequence of Definitions 2 and 7.

[3] A PFSM is observable if and only if the corresponding FA is deterministic.

Proposition 4 *If input sequence $\bar{x}$ distinguishes states s and s' of PFSM M then there is some $\bar{a} \in A^*$ such that $\bar{a}$ distinguishes states s and s' of PFA $F_P(M)$ and $in(\bar{a}) = \bar{x}$.*

Proposition 5 *Given PFSM M and input sequence $\bar{x}$, if there is some $\bar{a} \in A^*$ such that $\bar{a}$ distinguishes states s and s' of $F_P(M)$, no proper prefix of $\bar{a}$ distinguishes s and s', and $in(\bar{a}) = \bar{x}$ then $\bar{x}$ distinguishes states s and s' of M.*

Proposition 6 *There is an algorithm running in time $O((n_1 + n_2)^4)$ that takes as input two PFSMs M_1 and M_2 and determines whether M_1 and M_2 are equivalent, where n_1 and n_2 are the number of states of M_1 and M_2 respectively. If M_1 and M_2 are not equivalent then the algorithm outputs the lexicographically minimum input/output sequence for which M_1 and M_2 have different probabilities. This input/output sequence will always be of length at most $n_1 + n_2 + 1$.*

Proof:

This follows by applying Theorem 1 to $F_P(M)$ and $F_P(M')$.

Since our PFSM M is minimal, and so its states are pairwise distinguishable, we can define a set of input sequences that distinguish between the states of M.

Definition 8 *A set W of input sequences is a characterization set for PFSM M if for every pair (s, s') of states of M with $s \neq s'$ there exists some input sequence $\bar{x} \in W$ that distinguishes s and s'.*

The proof of the following is similar to that for the equivalent result for DFSMs (see, for example, [13]).

Proposition 7 *Let us suppose that M is a minimal OPFSM with n states. Then there exists a characterization set W for M with at most $n - 1$ input sequences where each sequence has length at most $n - 1$.*

The more general case, where M need not be observable, is similar.

Proposition 8 *Let us suppose that M is a minimal PFSM with n states. Then there exists a characterization set W for M with at most $n - 1$ input sequences where each sequence has length at most $2n + 1$.*

Proof:

States s and s' of M are distinguished by an input sequence $\bar{x}$ if and only if $\bar{x}$ distinguishes between the PFSM formed by changing the initial state of M to s and the PFSM formed by changing the initial state of M to s'. Thus, by Proposition 6, there is a sequence of length at most $2n + 1$ that distinguishes between any pair of distinct states of M.

Let $W = \{\bar{x}_1, \ldots \bar{x}_k\}$ denote a characterization set for M such that no proper subset of W is a characterization set for M. Let $\sim_i$ $(0 \leq i \leq k)$ denote the equivalence relation on the states of M such that: $s \sim_i s'$ if for all $1 \leq j \leq i$ we have that $\bar{x}_j$ does not distinguish between s and s'. Clearly $\sim_1$ has at least two equivalence classes and by the minimality of W we must have that for all $1 \leq i < k$, $\sim_{i+1}$ has more equivalence classes than $\sim_i$. Thus, the number of equivalence classes for $\sim_i$ $(1 \leq i \leq k)$ must be at least $i + 1$. However, the number of equivalence classes is bounded above by n and thus $k + 1 \leq n$ and so $k \leq n - 1$ as required.

Similar to testing from a DFSM [20], when identifying a given state s_i of M it may be sufficient to use a set of prefixes of sequences in W. Such a set is called an identification set.

Definition 9 *Given state s_i of minimal PFSM M and characterization set W, a set $W_i \subseteq Pre(W)$ is an* identification set *if for every state s_j of M with $s_i \neq s_j$, there is some input sequence $\bar{x} \in W_i$ that distinguishes s_i and s_j.*

4 Mutation operators

This section describes mutation operators for PFSMs and approaches to finding input sequences to distinguish the resultant mutants. Section 5 explains how testing can proceed on the basis of this. Throughout this section $M = (S, s_0, X, Y, h)$ denotes the PFSM being mutated and M' denotes a mutant. Proposition 6 tell us that we can decide whether M and M' are equivalent in time $O(n^4)$ and, if they are not equivalent, produce a sequence of length at most $2n + 1$ to distinguish them. In this section we consider conditions under which we can improve on this.

4.1 Changing the initial state

We can form a mutant of M by making some state $s \in S \setminus \{s_0\}$ of M the initial state and this gives $n - 1$ different mutants. Let $M_s = (S, s, X, Y, h)$ $(s \neq s_0)$ be such a mutant. Then we want to find a sequence that distinguishes the initial states of M and M_s. This is equivalent to the problem of finding a sequence to distinguish states s_0 and s of M. The following result is thus clear.

Proposition 9 *Let $M = (S, s_0, X, Y, h)$ be a PFSM and let $M_s = (S, s, X, Y, h)$ for some state $s \neq s_0$. If W_0 is an identification set for the initial state of M then there exists an input sequence $\bar{x} \in W_0$ that distinguishes M and M_s.*

Thus, any identification set W_0 for the initial state of M distinguishes M from every mutant of the form M_s for $s \neq s_0$. As shown in Section 3, $|W_0| \leq n-1$ and if M is observable then the elements of W_0 have length at most $n-1$ and otherwise they have length at most $2n+1$.

4.2 Altering probabilities

Suppose that $t = (s, x, y, s', p)$ is a transition of M and let Δ be a (possibly negative) value such that $0 \leq p + \Delta \leq 1$. We can mutate M by changing the probability associated with t to $p + \Delta$. Naturally, we must change the probability of at least one other transition from s with input x so that the sum of the probabilities is still 1. Let M' be a PFSM formed by altering the probability of t to $p' \neq p$.

Let us suppose that the probability of producing output y from state s of M in response to x is different from the probability of producing output y from state s of M' in response to x (this must be the case if M is observable). Then to distinguish M and M' it is sufficient to devise a sequence $\bar{a}$ in the following way. First, find a shortest path $\bar{a}_1$ in $F(M)$ from s_0 to s. Then we set $\bar{a} = \bar{a}_1(x, y, p)$.

Proposition 10 *Let $M(t, \Delta)$ be an PFSM formed by altering the probability of transition $t = (s, x, y, s', p)$ of PFSM M to $0 \leq p + \Delta \leq 1$ and suppose that the probability of producing output y from state s of M in response to x is different from the probability of producing output y from state s of M' in response to x (i.e. $p_M(s, x, y) \neq p_{M(t,\Delta)}(s, x, y)$). Let $\bar{a}_1$ be a shortest path in $F(M)$ from s_0 to s. If $\bar{a} = \bar{a}_1(x, y, p)$ then $in(\bar{a})$ distinguishes M and M' and has length at most n.*

Proof:

Let $\bar{x}_1$ and $\bar{y}_1$ be the input and output sequences from $\bar{a}_1$ respectively. By the minimality of $\bar{a}_1$, no path in $F(M)$ from s_0 with label $\bar{a}_1$ contains the transition t. Thus, $p^*_M(s_0, \bar{x}_1, \bar{y}_1) = p^*_{M(t,\Delta)}(s_0, \bar{x}_1, \bar{y}_1)$. Thus, since $p_M(s, x, y) \neq p_{M(t,\Delta)}(s, x, y)$ and for every state $s' \neq s$ we have that $p_M(s', x, y) = p_{M(t,\Delta)}(s', x, y)$ we have that $p^*_M(s_0, \bar{x}x, \bar{y}y) \neq p^*_{M(t,\Delta)}(s_0, \bar{x}_1 x, \bar{y}_1 y)$ as required. Finally, since $\bar{a}_1$ is a shortest path from s_0 to s it has length at most $n-1$ (since there are no repeated states) and so $\bar{a}$ has length at most n.

If the probabilities of producing output y in response to x from state s of M and $M(t, \Delta)$ are the same then by Proposition 6 we can decide in $O(n^4)$ whether M and $M(t, \Delta)$ are equivalent and, if they are not, find a sequence of length at most $2n+1$ that distinguishes them.

4.3 Changing the target state of a transition

Suppose $t = (s, x, y, s', p)$ is a transition of M and let s'' denote a state of M ($s' \neq s''$). We can create a new PFSM called $M(t, s'')$, by changing the ending state of t to s''. The following are clear.

Proposition 11 *Let t denote a transition (s, x, y, s', p) of PFSM M and let s'' be a state of M with $s' \neq s''$. If M is observable then $M(t, s'')$ is observable.*

Proposition 12 *If M is observable then there is an $O(n^2)$ time algorithm that decides whether $M(t, s'')$ and M are equivalent and, if they are not equivalent, returns an input sequence of length at most $2n + 1$ that distinguishes them.*

Naturally, if M is not observable and M and $M(t, s'')$ are not equivalent then in $O(n^4)$ we can produce an input sequence $\bar{x}$, of length at most $2n + 1$, that distinguishes them.

4.4 Creating a new transition

Let us suppose that $t = (s, x, y, s', p)$ is a transition of M, let s'' denote a state of M, let y' denote an output and let $p' < p$ denote a probability. We can create a new PFSM $M(t, s'', y', p')$, by reducing the probability associated with t to $p - p'$ and creating a new transition (s, x, y', s'', p'). If M has a transition from s to s'' with input x and output y' then we have simply altered probabilities and simulated a mutation operator already discussed in Subsection 4.2. Since this case is redundant we do not consider it here.

If $y' \neq y$ then we have reduced the probability of producing output y in response to x from s. Thus, we can distinguish M and $M(t, s'', y', p')$ by choosing a minimum length sequence $\bar{a}_1 \in A^*$ that labels a path in $F(M)$ from s_0 to s and use $in(\bar{a}_1)x$. This input sequence has length at most n. Otherwise we can refer to the result that we can decide in $O(n^4)$ time whether M and $M(t, s'', y', p')$ are equivalent and, if they are not, produce an input sequence of length at most $2n + 1$ that distinguishes them.

Example 2 *Figure 2 shows two mutants of the PFSM from Figure 1. The first is formed by changing the probability associated with transition $(s_3, i_1, o_1, s_3, \frac{1}{2})$ to $\frac{1}{4}$. Since the sum of probabilities of the transitions from s_3 with input i_1 must be 1, we also alter the probability attached to transition $(s_3, i_1, o_3, s_2, \frac{1}{2})$ to $\frac{3}{4}$. The second mutant is obtained from by adding a new transition $(s_2, i_3, o_3, s_3, \frac{2}{5})$. This forces us to decrease the probability associated to the transition $(s_2, i_3, o_1, s_2, 1)$ to $\frac{3}{5}$.*

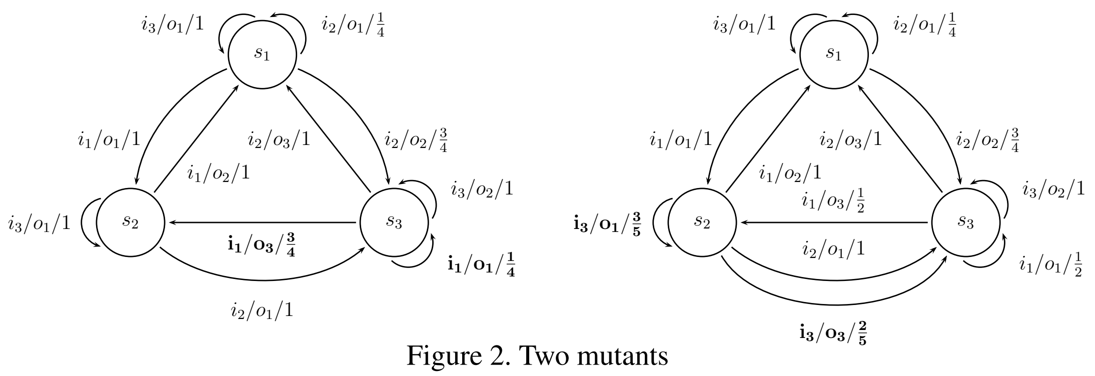


Figure 2. Two mutants

5 Applying the test sequences

Let M denote the specification PFSM, M' a mutant of M, and $\bar{x}/\bar{y}$ an input/output sequence such that $p^*_M(s_0, \bar{x}, \bar{y}) \neq p^*_{M'}(s'_0, \bar{x}, \bar{y})$ and so $\bar{x}$ distinguishes between M and M'. Let p_s denote $p^*_M(s_0, \bar{x}, \bar{y})$ and let p_m denote $p^*_{M'}(s'_0, \bar{x}, \bar{y})$. If we can determine the probability p of observing $\bar{y}$ in response to $\bar{x}$ in the IUT then we have two cases: if $p = p_s$ then the IUT is distinguished from the mutant M' and otherwise the IUT is faulty. However, we cannot determine p through testing; the best we can do is to produce an estimate $\hat{p}$ of p.

If we test the IUT with $\bar{x}$ a total of r times and in k of these tests we observe $\bar{y}$ then our estimate is $\hat{p} = \frac{k}{r}$. Naturally, the greater the value of r the higher our confidence in $\hat{p}$ being close to the true value p. We now show how statistical results regarding confidence intervals can be used in order to determine the required value of r.

Suppose that we fix a confidence level $c \in (0, 1)$ and we have this confidence of $\hat{p}$ being within e of p. The confidence denotes the probability that our estimate $\hat{p}$ satisfies $p - e < \hat{p} < p + e$. We want the estimate to either provide evidence that the IUT is faulty or that the IUT is not equivalent to M'. We are guaranteed to achieve this if we cannot have both p_s and p_m in the interval $(\hat{p} - e, \hat{p} + e)$. This is the case if $2e \leq |p_s - p_m|$ and thus we can set $e = \frac{|p_s - p_m|}{2}$. Naturally, we can choose a smaller value of e if we wish to have an estimate $\hat{p}$ with a smaller confidence interval.

Each application of $\bar{x}$ has two possible results: either the output sequence is $\bar{y}$ or it is some $\bar{y}' \neq \bar{y}$. We thus have a binomial distribution. We now discuss two sets of standard statistical results for binomial distributions that show how we can choose r given e and c.

Note that an alternative approach is to use hypothesis testing, with the null hypothesis either being that the true probability is the same as the probability in the mutant ($p = p_m$) or that the true probability is either p_m or 'on the other side of p_m from p_s' (i.e. if $p_m > p_s$ then the null hypothesis is $p - p_s \geq p_m - p_s$). However, here we focus on the use of confidence intervals since they have the additional benefit of allowing us to state the confidence we have of the true probability p being within e of the sample probability $\hat{p}$.

5.1 *Estimating with a small sample*

Since we have a binomial distribution with probability p, the probability of observing $\bar{y}$ in response to $\bar{x}$ a total of k times in r trials is

$$P(k, p, r) = \begin{pmatrix} r \\ k \end{pmatrix} p^k (1-p)^{(r-k)}$$

where

$$\begin{pmatrix} r \\ k \end{pmatrix} = \frac{r!}{k!(r-k)!}$$

Based on this we get the following likelihood function $L(p', k, r)$, which represents the probability that p' is the true probability if we have observed $\bar{y}$ a total of k times out of r tests with $\bar{x}$.

$$L(p', k, r) = \begin{pmatrix} r \\ k \end{pmatrix} p'^k (1-p')^{(r-k)}$$

Given k and r it is possible to calculate a value e such that we have confidence of at least c that $\hat{p}$ is within e of p and thus to determine whether we have tested a sufficient number of times. However, this computation becomes increasingly expensive as r increases and this motivates an interest in alternative approaches for large samples.

5.2 *Estimating with a large sample*

We can treat the mean $\bar{p}$ as a normally distributed random variable if $rp > 5$ and $r(1-p) > 5$ and this distribution has mean p and the following standard deviation (see, for example, [29]).

$$\sqrt{\frac{p(1-p)}{r}}$$

While we do not know p, we can check that $r\hat{p} > 5$ and $r(1-\hat{p}) > 5$ once the estimate has been produced. Here we consider this case.

Given a normal distribution with standard deviation σ, true mean μ and confidence value c there is a value z such that the proportion of the distribution that is within the region $(\mu - z\sigma, \mu + z\sigma)$ is c. For a confidence of 0.95 we can choose the value $z_0 = 1.96$ (see, for example, [29]). Clearly, we should choose r such that $z_0\sigma \leq e$. Since $\sigma = \sqrt{\frac{p(1-p)}{r}}$ we choose r such that

$$\sqrt{\frac{p(1-p)}{r}}z_0 \leq e$$

This can be rewritten to:

$$r \geq \frac{z_0^2 p(1-p)}{e^2}$$

Thus, we apply $\bar{x}$ a total of r times for some r that satisfies the above equation. While we do not know p in advance, we know that the worst case is with $p = \frac{1}{2}$ giving:

$$r \geq \frac{z_0^2}{4e^2}$$

If we require a value of $c = 0.95$, z_0 is slightly less than 2 and so it is sufficient to choose r such that:

$$r \geq \frac{1}{e^2}$$

If $\hat{p}$ is within e of the specified value p_s then we have the required confidence that the IUT is not equivalent to mutant M'; otherwise we have confidence that the IUT is faulty. Naturally, at this point we may wish to test the IUT further with $\bar{x}$ to gain an estimate with a narrower associated confidence interval.

6 Probabilistic-Stochastic Finite State Machines

In this section we extend the PFSM formalism in order to deal with stochastic time. We use random variables to model the (stochastic) time outputs take to be executed. We will consider that the sample space, that is, the domain of random variables, is a set of numeric time values $Time$. Since this is a *generic* time domain, the specifier can choose whether the system will use a discrete/continuous time domain. We simply assume that $0 \in Time$.

Definition 10 *We denote by $\mathcal{V}$ the set of random variables ($\xi, \psi, \ldots$ range over $\mathcal{V}$). Let ξ be a random variable. We define its probability distribution function as the function $F_\xi : Time \longrightarrow [0, 1]$ such that $F_\xi(x) = P(\xi \leq x)$, where $P(\xi \leq x)$ is the probability that ξ assumes values less than or equal to x. Let $\xi, \xi' \in \mathcal{V}$ be random variables. We write $\xi \approx \xi'$ if for all $x \in Time$ we have $F_\xi(x) = F_{\xi'}(x)$, that is, the random variables are equally distributed. We will denote by θ the random variable which probability distribution function is defined by $F_\theta(x) = 1$ for all $x \in Time$.*

Given two random variables ξ and ψ we consider that $\xi + \psi$ denotes a random variable distributed as the addition of the two random variables ξ and ψ.

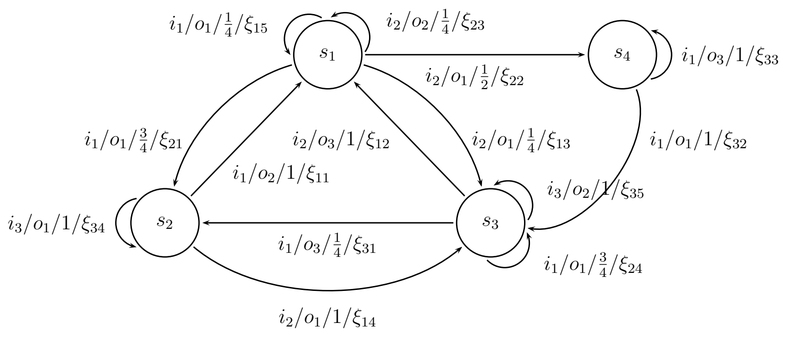
We will use the projection function π_i such that given a tuple $q = (q_1, \ldots, q_n)$, for all $1 \leq i \leq n$ we have $\pi_i(q) = q_i$.

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

Definition 11 *A Probabilistic Stochastic Finite State Machine, in short PSFSM, is a tuple $M = (S, s_0, X, Y, h)$ where S is a finite set of states, with $s_0 \in S$ being the initial state, X and Y denote the finite input and output alphabets, respectively, and $h : S \times X \leftrightarrow S \times Y \times (0, 1] \times \mathcal{V}$ is the transition relation. For each state $s \in S$ and input $x \in X$, $h(s, x)$ denotes a set of tuples of the form (s', y, p, ξ) where $s' \in S$, $y \in Y$, $p \in (0, 1]$, and $\xi \in \mathcal{V}$. For all $s \in S$ and $x \in X$, $\sum_{(s',y,p,\xi)\in h(s,x)} p = 1$.*

A tuple (s, x, y, p, ξ, s') is a transition of the machine if $(s', y, p, \xi) \in h(s, x)$, where $s, s' \in S$ are the initial and final states, $x \in X$ and $y \in Y$ are the input and output, p is the probability associated to the transition and $\xi \in \mathcal{V}$ is the random variable defining the time associated with the transition. Intuitively, a transition (s, x, y, p, ξ, s') indicates that if the machine is in state s and receives the input x then with probability p the machine emits the output y and it moves to state s' before time t with probability $F_\xi(t)$.

As we did when we defined PFSMs we do not allow a PSFSM to have two transitions $(s, x, y, p_1, \xi_1, s'), (s, x, y, p_2, \xi_2, s')$, that is, two transitions with the same initial and final states s, s' and the same input/output x/y. This constraint simplifies some computations, while it does not restrict the expressivity of our formalism. In fact, this condition does not limit the behaviours that we can define since the two previous transitions they have the same meaning as the one provided by a unique transition (s, x, y, p, ξ, s') where $p = p_1 + p_2$ and $\xi = \frac{p_1}{p} \cdot \xi_1 + \frac{p_2}{p} \cdot \xi_2$. We will explain later the meaning of the previous *normalization* of probabilities.


$i_1/o_1/\frac{1}{4}/\xi_{15}$
$i_2/o_2/\frac{1}{4}/\xi_{23}$
s_1
s_4
$i_1/o_3/1/\xi_{33}$
$i_2/o_1/\frac{1}{2}/\xi_{22}$
$i_1/o_1/\frac{3}{4}/\xi_{21}$
$i_2/o_3/1/\xi_{12}$
$i_2/o_1/\frac{1}{4}/\xi_{13}$
$i_1/o_1/1/\xi_{32}$
$i_1/o_2/1/\xi_{11}$
$i_3/o_2/1/\xi_{35}$
$i_3/o_1/1/\xi_{34}$
s_2
s_3
$i_1/o_3/\frac{1}{4}/\xi_{31}$
$i_1/o_1/\frac{3}{4}/\xi_{24}$
$i_2/o_1/1/\xi_{14}$
Figure 3. Probabilistic-Stochastic Finite State Machine.

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

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

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

for all $1 \leq i \leq 5$.

$$F_{\xi_{2i}}(x) = \begin{cases} 0 & \text{if } x < 4 \\ 1 & \text{if } x \geq 4 \end{cases}$$

for all $1 \leq i \leq 4$.

$$F_{\xi_{3i}}(x) = \begin{cases} 1 - e^{-2 \cdot x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

for all $1 \leq i \leq 5$.

Let us consider the transition $(s_3, i_2, o_3, 1, \xi_{12}, s_1)$*. Intuitively, if the machine is in state* s_3 *and receives the input* i_2 *then it will produce the output* o_3 *after a time given by* ξ_{12}*. For example, we know that this time will be less than* 1 *time unit with probability* $\frac{1}{5}$*, it will be less than* 3 *time units with probability* $\frac{3}{5}$*, and so on. Finally, once* 5 *time units have passed we know that the output has been performed (that is, we have probability* 1*).*

In the same way as the transitions of a PFSM can be characterized through a function p_M, an alternative characterization of a PSFSM M can be given by means of a suitable function providing the probability and the random variable associated to a transition.

Definition 12 *Let* $M = (S, s_0, X, Y, h)$ *be a PSFSM. We define the function* $\chi_M : S \times X \times Y \times S \to [0,1] \times \mathcal{V}$ *as:*

$$\chi_M(s, x, y, s') = \begin{cases} (p, \xi) & \text{if } \exists\, p, \xi : (s', y, p, \xi) \in h(s, x) \\ (0, \theta) & \text{otherwise} \end{cases}$$

Intuitively, $\chi_M(s, x, y, s')$ returns a pair (p, ξ) where p represents the probability that M moves to state s' and produces output y if it receives input x when in state s (that is, $p_M(s, x, y, s')$) and ξ corresponds to the random variable defining the time the system spends to perform the output y. If there does not exist such a transition then we return probability 0. In this case, the returned random variable can be discarded; we simply return the identity element θ. The relation between χ_M and h is given by $(s', y, p, \xi) \in h(s,x)$ if and only if $p > 0 \wedge \chi_M(s,x,y,s') = (p,\xi)$.

Next, we extend χ_M to sequences by means of a similar approach to the one introduced for extending p_M. First we give some additional notation.

Definition 13 *Given $\xi, \psi \in \mathcal{V}$ and $p, q \in [0,1]$ we define the combination of the pairs (p,ξ) and (q,ψ), denoted by $(p,\xi) \odot (q,\psi)$, as the pair $(p \cdot q, \xi + \psi)$.*

Given $\xi_i \in \mathcal{V}$ and $p_i \in [0,1]$, for all $1 \leq i \leq n$, we define the weighted addition of the pairs (p_i, ξ_i), denoted by $\bigoplus_{1 \leq j \leq n}(p_j, \xi_j)$, as the pair $\left(p, \sum_{1\leq j \leq n} \frac{p_j}{p} \cdot \xi_j\right)$ where $p = \sum_{1 \leq j \leq n} p_j > 0$; if $\sum_{1 \leq j \leq n} p_j = 0$ then we simply return $(0,\theta)$.

Let $M = (S, s_0, X, Y, h)$ be a PSFSM. We inductively define the function $\chi^*_M : S \times X^* \times Y^* \times S \longrightarrow [0,1] \times \mathcal{V}$ as

$$\chi^*_M(s,\epsilon,y,s') = \begin{cases} (1,\theta) & \text{if } y = \epsilon \text{ and } s' = s, \\ (0,\theta) & \text{otherwise} \end{cases}$$
$$\chi^*_M(s,\bar{x}x,\bar{y}y,s') = \bigoplus_{s'' \in S} \chi^*_M(s,\bar{x},\bar{y},s'') \odot \chi_M(s'',x,y,s')$$

Let us note that $\chi_M(s,\bar{x},\bar{y},s')$ is $(0,\theta)$ whenever $|\bar{x}| \neq |\bar{y}|$. Regarding the definition of $\bigoplus$, the second component of the pair represents the normalization of the random variables associated with the transitions of the sequences. Let us consider the machine depicted in Figure 3. Let us suppose the machine is in the initial state s_1 and it receives the input i_2. There are three transitions outgoing from s_1 with input i_2. So, as we explained, the machine will choose which one to perform according to the associated probabilities. Now, let us assume the output o_1 is the one selected. Then, we have two possible transitions labelled with i_2/o_1 and attached random variables ξ_{13} and ξ_{22}. The random variable that define the expected time the machine will spend to perform i_2/o_1 is given by $\frac{\frac{1}{2}}{\frac{1}{2}+\frac{1}{4}} \cdot \xi_{22} + \frac{\frac{1}{4}}{\frac{1}{2}+\frac{1}{4}} \cdot \xi_{13}$. In this way, we are *normalizing* the probabilities associated with the transitions that could be performed, since the other transition labelled with i_2 has been excluded in the pre-selection.

We will denote by $\chi^*_M(s,\bar{x},\bar{y})$ the pair (p,ξ) where p represents the probability that M produces output sequence $\bar{y}$ if it receives input sequence $\bar{x}$ when in state s and ξ the random variable that define the time the system spends in performing it. Thus, $\chi^*_M(s,\bar{x},\bar{y}) = \sum_{s' \in S} \chi^*_M(s,\bar{x},\bar{y},s')$. We can now extend the transition relation h to input sequences.

$$h^*(s,\bar{x}) = \{(s',\bar{y},p,\xi) | \chi^*_M(s,\bar{x},\bar{y},s') = (p,\xi) \wedge p > 0\}$$

During the rest of the paper, we assume that PSFSMs are completely specified. We consider both observable PSFSMs (OPSFSMs) and PSFSMs that are not observable as in the case of PFSMs. The concept of observability for PSFSM matches the definition presented in the previous formalism.

Regarding equivalence of states, we need to consider the stochastic component of our new model, so we extend the definition of equivalence of states given for PFSM. For two states of an PSFSM to be equivalent we need that they define the same sets of behaviours with the same probabilities and random variables. Thus, states s and s' of PSFSM M are *equivalent* if for every input/output sequence $\bar{x}/\bar{y}$, we have that $\chi^*_M(s, \bar{x}, \bar{y}) = \chi^*_M(s', \bar{x}, \bar{y})$. If s and s' are not equivalent then they are *distinguishable*.

7 Distinguishing states of a PSFSM

In this section we show how we can adapt and apply results from automata theory to produce sequences that allow us to distinguish two states of PSFSM M. Following the same structure applied to the previous formalism, PFSMs, we first consider the case where M is observable and then consider the general case where M may not be observable.

Definition 14 *Let $M = (S, s_0, X, Y, h)$ be a PSFSM. We define a FA $F_\chi(M) = (S, s_0, A, \delta, S)$ where A is $X \times Y \times (0, 1] \times \mathcal{V}$ and given $s \in S$, $x \in X$, $y \in Y$, $p \in (0, 1]$, and $\xi \in \mathcal{V}$ we have $\delta(s, (x, y, p, \xi)) = s'$ if and only if $(s', y, p, \xi) \in h(s, x)$.*

The following result shows that algorithms producing sequences that distinguish states of a FA can also be used to produce sequences that distinguish states of an OPSFSM.

Proposition 13 *If input sequence $\bar{x}$ distinguishes states s and s' of OPSFSM M then there is some $\bar{a} \in A^*$ such that $\bar{a}$ distinguishes states s and s' of $F_\chi(M)$ and $in(\bar{a}) = \bar{x}$.*

Proposition 14 *Given an OPSFSM M and $\bar{a} \in A^*$ such that $\bar{a}$ distinguishes states s and s' of $F_\chi(M)$, if no proper prefix of $\bar{a}$ distinguishes s and s' and $in(\bar{a}) = \bar{x}$ then $\bar{x}$ distinguishes states s and s' of M.*

The proof of the previous results is similar to the one corresponding to respectively, considering the definition of distinguishable states for PSFSMs the function χ^*_M, and the associated automata $F_\chi(M)$.

Proposition 15 *There is an algorithm running in time $O(n^2)$ that takes two states s_1 and s_2 of an OPSFSM M and determines whether they are equivalent, where n*

is the number of states of M. If s_1 and s_2 are distinguishable then the algorithm returns a minimal input/output sequence of length less than n that distinguishes them.

Proof: Since $F_\chi(M)$ is a minimal FA, there exists a set of sequences of length at most $n-1$ that pairwise distinguish the states of $F_\chi(M)$ and such a set can be found in $O(n^2)$ time (see, for example, [21]). The result thus follows from Proposition 14.

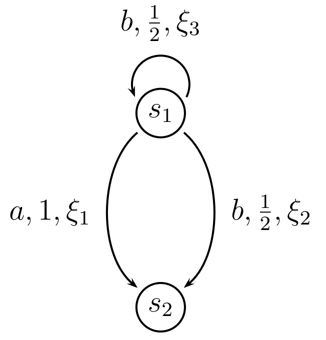
Next, we present an algorithm for determining if two probabilistic-stochastic finite automata, either observable or not, are equivalent and if they are not then the algorithm will output the lexicographically minimum string which the two automata will accept with different random variables. This algorithm is an adaptation of the result proved for probabilistic FA [56] and that was presented in Theorem 1. Next, we introduce the notation that will be used.

Definition 15 *A Probabilistic-Stochastic Finite Automaton (PSFA) N is defined by a tuple $(S, s_0, A, \delta, S_F, ps)$ in which S is a finite set of states, $s_0 \in S$ is the initial state, A is the finite alphabet, $\delta : S \times A \leftrightarrow S$ is the state transfer relation, $S_F \subseteq S$ is the set of final states, and ps is the transition probability-stochastic relation of type $S \times A \times S \leftrightarrow [0,1] \times \mathcal{V}$. If N receives $a \in A$ when in state $s \in S$ it moves to a state $s' \in \delta(s,a)$ with probability $\pi_1(ps(s,a,s'))$ and the time the system spends to perform the action is given by $\pi_2(ps(s,a,s'))$. The relation δ can be extended to take sequences from A^*, in the usual way.*

Let $Tr = \{(s,a,s') \mid \delta(s,a) = s'\}$ be the finite set of transitions of N and $K = \{1, 2, ..., |Tr|\}$. We consider that any bijection $f : K \longrightarrow Tr$ is an enumeration of Tr. We denote by Ψ_N the $|Tr|+1$-dimensional vector defined as $\Psi_N[1] = \theta$ and $\Psi_N[i+1] = \pi_2(ps(f(i)))$ for all $1 \leq i \leq |Tr|$.

We say that $(p, (q_1 \ldots, q_r))$ is an r-stochastic pair if $p \in [0,1]$ and for all $1 \leq i \leq r$ we have $q_i \in [0,1]$. We denote by $(0, 0_r)$ the pair where 0_r is the r-dimensional zero vector. We say that a matrix is stochastic if all its entries are r-stochastic pairs. Let $\mathcal{M}(i,j)$ be the set of all ($i \times j$)-dimensional stochastic matrices. We define the stochastic distribution function of N, $D_N : A \longrightarrow \mathcal{M}(n,n)$, as $D_N(a)[i,j] = (p, (q_1, \ldots, q_{m+1}))$ for all $a \in A$, where $n = |S|$, $m = |Tr|$, $p = \sum_{k=1}^{m+1} q_k$, $q_1 = 0$, and for all $1 \leq k \leq m$ we have $q_{k+1} = \pi_1(ps(f(k)))$ with $f(k) = (s_i, a, s_j)$.

Intuitively, a stochastic distribution function provides us with a stochastic matrix for each action in A. Each entry $D_N(a)[i,j]$ in the matrix corresponds to a stochastic pair that represents, on the one hand, the probability that the machine goes from state s_i to s_j with the corresponding action, that is, the probability associated to the transition (s_i, a, s_j), and in the other hand, the time the machine needs for performing the action from state s_i to state s_j. This time is defined by a random variable represented by a tuple $(q_1, q_2, \ldots, q_m)$. Each position, but the first one, represents a random variable that appears in the automaton, that is, the position i corresponds



$$
\begin{aligned}
f(1) &= (s_1, a, s_2)\\
f(2) &= (s_1, b, s_2)\\
f(3) &= (s_1, b, s_1)
\end{aligned}
$$

$$
D_N(a) = \begin{bmatrix} (0,(0,0,0)) & (1,(1,0,0)) \\ (0,(0,0,0)) & (0,(0,0,0)) \end{bmatrix}
$$

$$
D_N(b) = \begin{bmatrix} (\frac{1}{2},(0,0,\frac{1}{2})) & (\frac{1}{2},(0,\frac{1}{2},0)) \\ (0,(0,0,0)) & (0,(0,0,0)) \end{bmatrix}
$$

Figure 4. Example of Stochastic Matrices.

to the random variable labelling the transition $f(i-1)$; the first position is reserved to θ. Regarding the values, only one position, corresponding to the random variable attached to the transition (s_i, a, s_j), has a value different to 0 and equals to the probability associated to it.

Example 4 *Let us consider the PSFA depicted in Figure 4 and the stochastic matrices $D_N(a)$ and $D_N(b)$ corresponding to each of the actions belonging to the alphabet of the machine, that is, a and b. The matrices tell us if the machine can go from a state to another by performing the corresponding action. For example, on one the hand $D_N(b)[2,1] = D_N(b)[2,2] = (0,(0,0,0))$ denotes that the machine cannot perform the action b when in state s_2. On the other hand, $D_N(b)[1,1] = (\frac{1}{2},(0,0,\frac{1}{2}))$ means that the machine can move from state s_1 to s_1 by performing b. In addition, these matrices provide the random variables that define the time that the system spends in performing the corresponding action.*

Next, we extend the function D_N to sequences. First we introduce additional notation we require in order to define it.

Definition 16 *Given two stochastic pairs $v = (p,(v_1,\ldots,v_r))$ and $u = (q,(u_1,\ldots,u_r))$ we define the combination of v and u as:*

$$v \bowtie u = \begin{cases} (0, 0_r) & \text{if } u = (0, 0_r) \vee v = (0, 0_r) \\ (p \cdot q, q \cdot (v_1, \ldots, v_r) + p \cdot (u_1, \ldots, u_r)) & \text{otherwise} \end{cases}$$

Given $Q_1 \in \mathcal{M}(p, q)$ and $Q_2 \in \mathcal{M}(q, r)$ we define the combination of these stochastic matrices, denoted by $Q_1 \otimes Q_2$, as the matrix where for all $1 \leq i \leq p$ and all $1 \leq j \leq r$ we have

$$Q_1 \otimes Q_2[i, j] = \sum_{k=1}^{q} Q_1[i, k] \bowtie Q_2[k, j]$$

Let $N = (S, s_0, A, \delta, S, ps)$ be a PSFA, $\bar{a} \in A^*$, and $a \in A$, we define the function $D_N^* : A^* \longrightarrow \mathcal{M}(n, n)$ in the following way:

$$\begin{aligned} D_N^*(\epsilon) &= \mathrm{E_n} \\ D_N^*(\bar{a}a) &= D_N^*(\bar{a}) \otimes D_N(a) \end{aligned}$$

where $\mathrm{E_n}$ is an n-dimensional stochastic matrix defined as:

$$\mathrm{E_n}[i, j] = \begin{cases} (1, 0_m) \text{ if } i = j \\ (0, 0_m) \text{ if } i \neq j \end{cases}$$

where $m = |Tr| + 1$.

In order to establish the equivalence of two PSFAs we need to determine the equivalence of their initial states. Thus, we will deal with the sequences outgoing from the initial states of the automata whose random variables appear in the first row of the corresponding stochastic matrices. In addition, we only consider those entries of the matrix corresponding to a final state.

Definition 17 *Let $N = (S, s_0, A, \delta, S_F, ps)$ be a PSFA having n states. We define the vector $U(\bar{a}) = \lambda_N \otimes D_N(\bar{a})$, where λ_N is an n-dimensional vector of m-stochastic pairs, with $\lambda_N(1) = (1, 0_m)$ and for all $2 \leq i \leq n$ $\lambda_N(i) = (0, 0_m)$, with $m = |Tr| + 1$.*

The vector of accepting probabilities and random variables of a sequence $\bar{a}$ by N, denoted by $P(\bar{a})$, is given by $U(\bar{a}) \otimes F_n$, where F_n is the $n \times n$-dimensional matrix *defined as:*

$$F_n[i,j] = \begin{cases} (1, 0_m) \text{ if } s_i \in S_F \wedge i = j \\ \\ (0, 0_m) \text{ otherwise} \end{cases}$$

Let $X = ((p_1, (q_{11}, \ldots, q_{1m})), \ldots, (p_n, (q_{n1}, \ldots, q_{nm})))$ *be a vector of* m*-stochastic pairs. We denote by* X_{Ψ_N} *the vector defined as* $X_{\Psi_N}[i] = (\pi_1(X[i]), \pi_2(X[i]) \cdot (\Psi_N)^T)$ *for all* $1 \leq i \leq n$

The vector $U(\bar{a})$ represents the probabilities and random variables associated to the performance of the sequence $\bar{a}$ from the initial state. The matrix F_n allows us to consider only the probabilities and random variables when the automaton reaches a final state. Intuitively, the vector X_{Ψ_N} is a transformation of the vector X where the second component of the stochastic pairs are replaced by the random variable that they represent. That is, given the stochastic pair $X[i] = (p_i, (q_{i1}, \ldots, q_{im}))$ we will obtain $X_{\Psi_N}[i] = (p_i, q_{i1} \cdot \theta + q_{i2} \cdot \xi_1 \ldots + q_{im} \cdot \xi_{m-1})$, where $m-1$ is the number of transitions.

Next, we introduce the notion of equivalent PSFAs. Two PSFAs are equivalent if their initial states accept the same sequences with the same probability and the associated random variables are equally distributed.

Definition 18 *Let* $N_1 = (S_1, s_0^1, A, \delta_1, S_{F_1}, ps_1)$ *and* $N_2 = (S_2, s_0^2, A, \delta_2, S_{F_2}, ps_2)$ *be two PSFAs with* n_1 *and* n_2 *states respectively. Then,* N_1 *and* N_2 *are equivalent if for all* $\bar{a} \in A^*$*,* N_1 *and* N_2 *accept* $\bar{a}$ *with equal probabilities and equal random variables, that is,*

- $\Pi_1(\bigoplus_{i=1}^{n_1} P^1_{\Psi_{N_1}}(\bar{a})[i]) = \Pi_1(\bigoplus_{i=1}^{n_2} P^2_{\Psi_{N_2}}(\bar{a})[i])$ *and*
- $\Pi_2(\bigoplus_{i=1}^{n_1} P^1_{\Psi_{N_1}}(\bar{a})[i]) \approx \Pi_2(\bigoplus_{i=1}^{n_2} P^2_{\Psi_{N_2}}(\bar{a})[i])$.

The operator $\bigoplus$ provides us the normalization of the random variables that define the time the automaton would spend to perform the sequence from the initial state and the probability of performing the sequence.

Next, we introduce the notation that we will use in order to establish the equivalence of two PSFAs. We extend the definition of the matrices D_N,D_N^*, and U.

Definition 19 *Let* $N_1 = (S_1, s_0^1, A, \delta_1, S_{F_1}, ps_1)$ *and* $N_2 = (S_2, s_0^2, A, \delta_2, S_{F_2}, ps_2)$ *be two PSFAs with* n_1 *and* n_2 *states respectively. For all* $a \in A$ *we define*

$$D_{N_1 \star N_2}(a) = \begin{bmatrix} D^*_{N_1}(a) & 0_{n_1 \times n_2} \\ 0_{n_2 \times n_1} & D^*_{N_2}(a) \end{bmatrix}$$

where $0_{r\times s}$ is the $(r \times s)$-dimensional zero matrix. Then, for all $a \in A$ and $\bar{a} \in A^*$, we have $D^*_{N_1\star N_2}(\bar{a}a) = D^*_{N_1\star N_2}(\bar{a}) \otimes D_{N_1\star N_2}(a)$. We also define, for all $\bar{a} \in A^*$, the vector $U_{N_1\star N_2}(\bar{a}) = [\lambda_{N_1}, \lambda_{N_2}] \otimes D^*_{N_1\star N_2}(\bar{a})$.

The multi-projection function Π^s_r is such that for all tuple $q = (q_1, \ldots, q_n)$, we have $\Pi^s_r(q) = (q_r, \ldots, q_s)$, for all $1 \leq r \leq s \leq n$. Finally, let span be the function that maps a set of vectors to the vector space generated by the vectors in the set. Then, we define the set $H(N_1, N_2) = \{U_{N_1\star N_2}(\bar{a}) | \bar{a} \in A^*\}$.

Lemma 1 *Let N_1 and N_2 be two PSFAs where n_1 and n_2 are the number of states of N_1 and N_2 respectively. If V is a basis for $span(H(N_1, N_2))$ then N_1 and N_2 are equivalent if and only if $\forall \vec{v} \in V$ we have*

- $\Pi_1(\bigoplus_{i=1}^{n_1}(\vec{v}^{\,1} \otimes F_{n1})_{\Psi_{N_1}}) = \Pi_1(\bigoplus_{i=1}^{n_2}(\vec{v}^{\,2} \otimes F_{n2})_{\Psi_{N_2}})$ *and*
- $\Pi_2(\bigoplus_{i=1}^{n_1}(\vec{v}^{\,1}_{\otimes} F_{n1})\Psi_{N_1}) \approx \Pi_2(\bigoplus_{i=1}^{n_2}(\vec{v}^{\,2} \otimes F_{n2})_{\Psi_{N_2}})$

where $\vec{v}^{\,1} = \Pi^{n_1}_1(\vec{v})$ and $\vec{v}^{\,2} = \Pi^{n2+n1}_{n_1+1}(\vec{v})$.

The proof of this result, taking into account that the notion of equivalence of two PSFAs N_1 and N_2 fits with the imposed requirements to the vectors in the basis, is straightforward.

Theorem 2 *There is an algorithm that takes two PSFAs N_1 and N_2 and determines whether N_1 and N_2 are equivalent, where n_1 and n_2 are the number of states of N_1 and N_2 respectively. If N_1 and N_2 are not equivalent then the algorithm outputs the lexicographically minimum string that is accepted by N_1 and N_2 with different probability or/and random variables. This string will always be of length at most $n_1 \cdot (|Tr_1| + 1) + n_2 \cdot (|Tr_2| + 1) - 1$.*

First, we present an algorithm that allows us to establish the equivalence of two PSFAs. The basic idea is to find a basis $V \subseteq H(N_1, N_2)$ for the vector space $span(H(N_1, N_2))$. Since the dimension of the vector space $span(H(N_1, N_2))$ is at most $n_1 \cdot (|Tr_1| + 1) + n_2 \cdot (|Tr_2| + 1)$, where Tr_1 and Tr_2 are the sets of transitions of N_1 and N_2, respectively, the number of elements in V is at most $n_1 \cdot (|Tr_1| + 1) + n_2 \cdot (|Tr_2| + 1)$.

Let us explain how the algorithm depicted in Figure 5 works. We define a tree T which has a node for every $\bar{a} \in A^*$. The root of T is $node(\epsilon)$. Every $node(\bar{a})$ in the tree T has $|A|$ children. Let $U_{N_1\star N_2}(\bar{a})$ be the vector associated with $node(\bar{a})$. The vector associated to node($\bar{a}a$), $U_{N_1\star N_2}(\bar{a}a)$, can be computed as $U_{N_1\star N_2}(\bar{a}) \otimes D_{N_1\star N_2}(a)$.

The method applied to determine whether N_1 and N_2 are equivalent is to prune the tree T. Initially, V is assigned the empty set. The nodes of the tree are visited

```
Input:

$N_1 = (S_1, s_0^1, A, \delta_1, S_{F_1}, ps_1)$
$N_2 = (S_2, s_0^2, A, \delta_2, S_{F_2}, ps_2)$.

Initialization:
• $Nodes := \emptyset; V := \emptyset; notEq := \emptyset$.
• $queue \leftarrow node(\epsilon)$.
• $equival := true$; $seq := \epsilon$
 (1) While $queue$ is not empty do
     (a) $queue \rightarrow node(\bar{a})$
     (b) If $U_{N_1 \star N_2}(\bar{a}) \notin span(V)$ then
           (i) For all $a \in A$: $queue \leftarrow node(\bar{a}a)$
          (ii) $V := V \cup \{U_{N_1 \star N_2}(\bar{a})\}$
         (iii) $Nodes := Nodes \cup node(\bar{a})$
     (c) For all $\vec{v} = U_{N_1 \star N_2}(\bar{a}) \in V$:
           (i) $\vec{v}^{\,1} = \Pi_1^{n_1}(\vec{v})$;
          (ii) $\vec{v}^{\,2} = \Pi_{n_1+1}^{n2+n1}(\vec{v})$;
         (iii) if $\Pi_1(\bigoplus_{i=1}^{n_1}(\vec{v}^{\,1} \otimes F_{n1})_{\Psi_{N_1}}) \neq \Pi_1(\bigoplus_{i=1}^{n_2}(\vec{v}^{\,2} \otimes F_{n2})_{\Psi_{N_2}}) \vee$
               $\Pi_2(\bigoplus_{i=1}^{n_1}(\vec{v}^{\,1}_{\otimes} F_{n1})\Psi_{N_1}) \not\approx \Pi_2(\bigoplus_{i=1}^{n_2}(\vec{v}^{\,2} \otimes F_{n2})_{\Psi_{N_2}})$ then
                 • $equival := false$;
                 • $notEq := notEq \cup \{\bar{a}\}$;
     (d) if not $equival$ then $seq := min\{\bar{a} \in notEq\}$
     (e) $return(equival, seq)$
```

Figure 5. Algorithm of equivalence of automata.

in breadth-first order. At each node $node(\bar{a})$, we check whether its associated vector, $U_{N_1 \star N_2}(\bar{a})$, is linearly independent with respect to the ones in V. If it is, the vector is included in the set V; otherwise, we prune the subtree rooted at $node(\bar{a})$. Additionally, we will use another set $Nodes$ where we add the nodes whose associated vector is selected and included in V. We stop traversing the tree when every node is either visited or pruned. The vectors in the resulting set V will be linearly independent and they are a basis for $span(H(N_1, N_2))$.

Finally, the algorithm finishes the inspection of the nodes, it verifies if the automata are equivalent. Otherwise, the algorithm returns the lexicographically minimum string in the set $\{\bar{a} \mid node(\bar{a}) \in Nodes\}$ which is accepted by N_1 and N_2 with different random variables or probabilities.

Proof: Let $T_{N_1 \star N_2}$ be the tree formed by the nodes in $Nodes \cup \{node(\bar{a}a) \mid \bar{a} \in Nodes \wedge a \in A\}$, that is, the set of nodes that have been visited. Because the vectors in V are $(n_1 \cdot (|Tr_1|+1) + n_2 \cdot (|Tr_2|+1))$-dimensional, $T_{N_1 \star N_2}$ has at most $n_1 \cdot (|Tr_1| + 1) + n_2 \cdot (|Tr_2| + 1)$ internal nodes ($|Nodes|$). The set V consists of the vectors associated with the internal nodes of $T_{N_1 \star N_2}$. Since we prune tree T at

$node(\bar{a})$ if $U_{N_1\star N_2}(\bar{a}) \in span(V)$, the vectors associated with the leaves of $T_{N_1\star N_2}$ will be linearly dependent to the vectors in V.

We will prove that the vectors in the resulting set V form a basis for the vector space $span(H(N_1, N_2))$. For $i \geq 0$, let $V_i = \{U_{N_1\star N_2}(\bar{a}\bar{b}) \mid node(\bar{a})$ is a leaf of $T_{N_1\star N_2} \wedge |\bar{b}| = i\}$. The set V_0 contains the vectors associated with the leaves of $T_{N_1\star N_2}$ and the set V_i, $i \geq 1$, is the set of the vectors associated with the unvisited nodes of T which are of distance i from a leaf. It can be seen that

$$span(V \cup \bigcup_{i=0}^{\infty} V_i) = span(\{U_{N_1\star N_2}(\bar{a}) : \bar{a} \in A^*\}) = span(H(N_1, N_2))$$

Lemma 2 *For all $i \geq 0$ we have $V_i \subseteq span(V)$.*

Proof: Let $V = \{\vec{v_1}, \ldots, \vec{v_r}\}$ where $r \leq n_1 \cdot (|Tr_1| + 1) + n_2 \cdot (|Tr_2| + 1)$. We prove this lemma by induction on i.

The base case $V_0 \subseteq \{\vec{v}\otimes D_{N_1\star N_2}(a) : \vec{v} \in V \wedge a \in A\} \subseteq span(V)$ follows from the algorithm. Let us assume that $V_i \subseteq span(V)$. Then, for all $\bar{a}, \bar{b} \in A^*$ and $c \in A$ such that node($\bar{a}$) is a leaf and $|\bar{b}| = i$ we have $U_{N_1\star N_2}(\bar{a}\bar{b}c) = U_{N_1\star N_2}(\bar{a}\bar{b})\otimes D_{N_1\star N_2}(c) = (\sum_{i=1}^{r} m_i\vec{v_i})\otimes D_{N_1\star N_2}(c) = \sum_{i=1}^{r} m_i(\vec{v_i}\otimes D_{N_1\star N_2}(c)) \in span(V \cup V_0) = span(V)$.

Thus, the vectors in V form a basis for $span(H(N_1, N_2))$. By Lemma 1, the algorithm can determine whether N_1 and N_2 are equivalent.

Lemma 3 *Let N_1 and N_2 be two PSFAs with n_1 and n_2 states respectively. If N_1 and N_2 are not equivalent then the algorithm depicted in Figure 5 outputs the lexicographically minimum string which is accepted by N_1 and N_2 with different probabilities or/and random variables. This string will be of length at most $n_1 \cdot (|Tr_1| + 1) + n_2 \cdot (|Tr_2| + 1) - 1$.*

Proof: Let str be the function that maps a string to its lexicographic order in A^*. Let $\bar{a}$ be the string returned by the algorithm. Let us assume that the result does not hold. Let $\bar{b} \in A^*$ be the lexicographically minimum string such that $str(\bar{b}) < str(\bar{a})$ and fulfills either $\Pi_1(\bigoplus_{i=1}^{n_1} P^1_{\Psi_{N_1}}(\bar{b})[i]) \neq \Pi_1(\bigoplus_{i=1}^{n_2} P^2_{\Psi_{N_2}}(\bar{b})[i])$ or $\Pi_2(\bigoplus_{i=1}^{n_1} P^1_{\Psi_{N_1}}(\bar{b})[i]) \not\approx \Pi_2(\bigoplus_{i=1}^{n_2} P^2_{\Psi_{N_2}}(\bar{b})[i])$. Since $node(\bar{b}) \notin Nodes$, there must exist a leaf $node(\bar{x})$ such that $\bar{b} = \bar{x}\bar{y}$ for some $\bar{y} \in A^*$. Since we used a breadth-first transversal in the algorithm, the vector $U_{N_1\star N_2}(\bar{x})$ will be in $span(\{U_{N_1\star N_2}(\bar{u}) \mid \bar{u} \in A^* \wedge str(\bar{u}) < str(\bar{x})\})$. If we consider the assumption regarding random variables we have

$$\Pi_2(\bigoplus_{i=1}^{n_1} P^1_{\Psi_{N_1}}(\bar{b})[i]) \not\approx \Pi_2(\bigoplus_{i=1}^{n_2} P^2_{\Psi_{N_2}}(\bar{b})[i])$$
$$\Updownarrow$$
$$\Pi_2(\bigoplus_{i=1}^{n_1}(P_{N_1}(\bar{x}) \otimes D^*_{N_1}(\bar{y}))_{\Psi_{N_1}})[i]) \not\approx \Pi_2(\bigoplus_{i=1}^{n_2}(P_{N_2}(\bar{x}) \otimes D^*_{N_2}(\bar{y}))_{\Psi_{N_2}})[i])$$
$$\Updownarrow$$
$$\Pi_2(\bigoplus_{i=1}^{n_1}(\sum_{str(\bar{u})<str(\bar{x})} m_{\bar{u}} \cdot P_{N_1}(\bar{u}) \otimes D^*_{N_1}(\bar{y}))_{\Psi_{N_1}})[i])$$
$$\not\approx \Pi_2(\bigoplus_{i=1}^{n_2}(\sum_{str(\bar{u})<str(\bar{x})} m_{\bar{u}} \cdot P_{N_2}(\bar{u}) \otimes D^*_{N_2}(\bar{y}))_{\Psi_{N_2}})[i])$$
$$\Updownarrow$$
$$\Pi_2(\bigoplus_{i=1}^{n_1}(\sum_{str(\bar{u})<str(\bar{x})} m_{\bar{u}} \cdot P_{N_1}(\bar{u}\bar{y})_{\Psi_{N_1}})[i]) \not\approx \Pi_2(\bigoplus_{i=1}^{n_2}(\sum_{str(\bar{u})<str(\bar{x})} m_{\bar{u}} \cdot P_{N_2}(\bar{u}\bar{y})_{\Psi_{N_2}})[i])$$
$$\Updownarrow$$
$$\Pi_2(\bigoplus_{i=1}^{n_1}(\sum_{str(\bar{u})<str(\bar{x})} m_{\bar{u}} \cdot P^1_{\Psi_{N_1}}(\bar{u}\bar{y}))[i]) \not\approx \Pi_2(\bigoplus_{i=1}^{n_2}(\sum_{str(\bar{u})<str(\bar{x})} m_{\bar{u}} \cdot P^2_{\Psi_{N_2}}(\bar{u}\bar{y}))[i])$$

Since $str(\bar{u}\bar{y}) < str(\bar{x}\bar{y}) = str(\bar{b})$ for all $str(\bar{u}) < str(\bar{x})$, the last inequality is false. It contradicts the assumption $\Pi_2(\bigoplus_{i=1}^{n_1} P^1_{\Psi_{N_1}}(\bar{b})[i]) \not\approx \Pi_2(\bigoplus_{i=1}^{n_2} P^2_{\Psi_{N_2}}(\bar{b})[i])$. The same reasoning is applied if we consider the assumption about the probabilities. Therefore the sequence $\bar{a}$ returned by the algorithm will be the lexicographically minimum string whose accepting random variables by N_1 and N_2 are different. Since no node in $Nodes$ is labelled by a sequence of length $> n_1 \cdot (|Tr_1| + 1) + n_2 \cdot (|Tr_2| + 1) - 1$, the length of sequence $\bar{a}$ will be at most $n_1 \cdot (|Tr_1| + 1) + n_2 \cdot (|Tr_2| + 1) - 1$.

We now show how this result can be applied to PSFSMs. First, we introduce the definition of the PSFA associated with a PSFSM.

Definition 20 *Given PSFSM $M = (S, s_0, X, Y, h)$ we can define the associated PSFA $F_S(M) = (S, s_0, A, \delta, S, ps)$ in which*

(1) $S_E = S \cup \{s_E\}$ where $s_E \notin S$
(2) The alphabet A is $X \times Y$.
(3) Given $s \in S$, $x \in X$, $y \in Y$, and $p \in (0, 1]$, $\delta(s, (x, y)) = s'$ and $ps(s, (x, y), s') = (p, \xi)$ if and only if $(s', y, p, \xi) \in h(s, x)$
(4) Given $s \in S$, $x \in X$, and $y \in Y$ we have $\delta(s, (x, y)) = s_E$ and $ps(s, (x, y), s_E) = (p, \theta)$ if and only if $p > 0$ and $p = 1 - \sum_{(s'', y, q, \xi) \in h(s,x)} q$.

As in the previous formalism, we need to adjust the probabilities when we define the associated PFA. In order to do it, we extend the set of states with a new one called s_E, and include new transition for reaching probability 1 for each state and label in the PSFA. In this case, we associate to these new transitions the random variable θ. We denote by Ex_M the set of transitions ending up in the state s_E.

Proposition 16 *There is an algorithm that takes as input two PSFSMs M_1 and M_2 and determines whether M_1 and M_2 are equivalent, where n_1 and n_2 are the number of states of M_1 and M_2, respectively. If M_1 and M_2 are not equivalent then the algorithm outputs the lexicographically minimum input/output sequence for which M_1 and M_2 have different probabilities or random variables. This input/output sequence will always be of length at most $n_1 \cdot (|Tr_1| + |Ex_{M_1}| + 1) + n_2 \cdot (|Tr_2| + |Ex_{M_2}| + 1) + 1$.*

Proof : The result follows by applying Theorem 2 to $F_P(M)$ and $F_P(M')$. Let us note that we need to consider the number of transitions that are added to the associated automata that end up in the error state. This affects the length of the input/output sequence produced by the algorithm, in case that M_1 and M_2 are not equivalent.

8 Mutation operators for PSFSM

This section describes a mutation operator for PSFSMs and approaches to finding input sequences to distinguish the resultant mutants. Since PSFSMs include probabilities we could apply the mutation operators described for PFSMs in Section 4 for obtaining mutants. We only need to apply the results presented in the previous section for the new formalism. The adaptation will be redundant and we do not consider it here. Proposition 16 tell us that we can decide whether M and M' are equivalent and, if they are not equivalent, produce a sequence to distinguish them. In this section we consider conditions under which we can improve this.

Let us consider a transition $t = (s, x, y, s', p, \xi)$ of M. We can mutate M by changing the random variable associated with t to ψ. Let M' be a PSFSM formed by altering the random variable of t to ψ.

Let us suppose that the random variable for performing output y from state s of M in response to x is different from the random variable for performing output y from state s of M' in response to x (this must be the case if M is observable). Then to distinguish M and M' it is sufficient to devise a sequence $\bar{a}$ in the following way. First, find a shortest path $\bar{a}_1$ in $F(M)$ from s_0 to s. Then, we set $\bar{a} = \bar{a}_1(x, y, p, \xi)$.

Proposition 17 *Let M_ξ be a PSFSM formed by replacing the random variable of a transition $t = (s, x, y, s', p, \xi)$ of PSFSM M by ψ and suppose that ξ and ψ are not equally distributed, that is, $\xi \not\approx \psi$. Let $\bar{a}_1$ be a shortest path in $F(M)$ from s_0 to s. If $\bar{a} = \bar{a}_1(x, y, p, \xi)$ then $in(\bar{a})$ distinguishes M and M_ξ and has length at most n.*

Proof : Let $\bar{x}_1$ and $\bar{y}_1$ be the input and output sequences from $\bar{a}_1$ respectively. By the minimality of $\bar{a}_1$, no path in $F(M)$ from s_0 with label $\bar{a}_1$ contains the transition t. Thus, $\chi^*_M(s_0, \bar{x}_1, \bar{y}_1) = \chi^*_{M_\xi}(s_0, \bar{x}_1, \bar{y}_1)$. Since $\chi_M(s, x, y) \neq \chi_{M_\xi}(s, x, y)$ and for every state $s' \neq s$ we have $\chi_M(s', x, y) = \chi_{M_\xi}(s', x, y)$ we conclude $\chi^*_M(s_0, \bar{x}x, \bar{y}y) \neq \chi^*_{M_\xi}(s_0, \bar{x}_1 x, \bar{y}_1 y)$, as required. Finally, since $\bar{a}_1$ is a shortest path from s_0 to s, it has length at most $n-1$ (since there are no repeated states) and so $\bar{a}$ has length at most n.

If M is not observable, by Proposition 16 we can decide whether M and M_ξ are equivalent and, if they are not, find a sequence of length at most $2 \cdot n + 2 \cdot (|Tr| + |Ex_M|) + 3$ that distinguishes them.

9 Related work

This section describes previous work on testing probabilistic systems. Interestingly, there has been relatively little work on this but there has been a considerable amount of work on model checking PFSMs (see for example [2,57,16,33,53]).

It has been noted that we can consider the testing of a stateless system to be a process of sampling its behaviour. If the test suite used is randomly generated, possibly based on a distribution (operational profile) that reflects expected usage then the result of testing can be used to estimate the reliability of the IUT [12,7]. Further, we can place a confidence in the observed reliability reflecting the true reliability of the IUT within some margin.

Researchers have tackled the problem of testing from an observable NFSM when each transition introduces a random delay and the expected delay for a transition is represented by a probability distribution [45]. The problem is to test to determine whether the distribution for each transition in the IUT is correct, where correctness is represented by a range of conformance relations. Testing is used to check that the IUT satisfies the conformance relation, relative to the specification, within a given degree of confidence.

It is sometimes desirable to have a process that can be applied once in order to take a conforming IUT to a given state or (strongly) distinguish two states of the specification and thus of a conforming IUT. Naturally, there need not exist single sequences that are guaranteed to achieve this and thus it is normal to apply an adaptive process. Alur et al. [1] show that it is PSPACE-complete to determine whether there is a single input sequence that strongly distinguishes two states of a PFSM and that it is EXPTIME-complete to determine whether there is an adaptive process that strongly distinguishes two states of a PFSM. Zhang and Zheung show how policies (adaptive processes) can be generated to move an OPFSM from state s to another state s' and how a policy can be found to (strongly) distinguish two states of an OPFSM [60]. Note that the work of Alur et al. [1] and Zhang and Zheung [60] refer to producing a single sequence or policy that will achieve the desired results through one application only. In contrast, we assume that there is a reliable reset operation and thus that we can *repeatedly* apply an input sequence or policy.

A related problem is machine identification; we wish to model the IUT rather than test to check that the IUT conforms to a given model. This problem has been considered in the context of probabilistic state machines [48]. In this approach, the set of observed traces is used to induce a FA in the classical way. The probabilities on each transition are then estimated by determining the ratios of the labels (from each state) observed in testing.

10 Conclusions

This paper developed mutation testing techniques for probabilistic finite state machines (PFSMs) and probabilistic-stochastic finite state machines (PSFSMs). It defined several mutation operators and adapted results from the theory of probabilistic finite automata in order to produce test sequences that distinguish a PFSM M from a mutant M'. An important property of PFSMs and PSFSMs is that given two PFSMs or PSFSMs M and M' we can decide in polynomial time whether M and M' are equivalent. As a result, mutation testing for PFSMs and PSFSMs does not suffer from the equivalent mutant problem.

An input sequence $\bar{x}$ kills a mutant M' if there is some output sequence $\bar{y}$ such that the probabilities of observing $\bar{x}/\bar{y}$ in M and M' are different. When $\bar{x}$ is used in testing we observe resultant input/output sequences and ideally we would like to compare the probabilities of each of these with those expected. However, we cannot determine the true probabilities through testing and so this paper has shown how results from statistic sampling theory can be used to estimate the probabilities with sufficient precision, up to a required level of confidence.

It is interesting to consider what faults the mutations represent and to thus define the corresponding fault model. Since we are assuming that the coupling effect holds, the fault model is the set of models that can be produced from our original model M using a sequence of zero or more mutations. The first observation is that if we have probabilities and not time then the fault model is all PFSM with the same input and output sets as M and no more states than M. When we include stochastic time, the fault model allows the distribution used to change. Naturally, there are a number of ways of introducing additional mutation operators that extend this fault model. For example, we might allow a mutation to change the end state of a transition to be a new state, although we would then have to define transitions from this new state. A natural way of doing this, for a transition t with end state s, is to produce a new state s' equivalent to s, change the end state of t to s', and then mutate one or more transition that having starting state s' using the mutation operators already defined. We could also allow a mutation operator to replace a distribution by another distribution, but to make this useful it would be important to make sure that the two distributions were ‘similar’ in important ways, such as having almost identical means.

We have shown how an input sequence can be efficiently generated to kill a mutant M' of M. However, in applying the resultant input sequences the number of repetitions depends on the probabilities of the corresponding input/output sequence in M and M'. There thus remains the following question: how can we produce an input sequence $\bar{x}$ that kills M' and minimizes the test execution effort for a given required confidence interval c?

Acknowledgements

This research was partially supported by the Spanish MEC project WEST/FAST TIN2006-15578-C02 and the Marie Curie RTN *TAROT* (MRTN-CT-2003-505121).

References

[1] R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *27th ACM Symp. on Theory of Computing, STOC’95*, pages 363–372. ACM Press, 1995.

[2] C. Baier, M.Z. Kwiatkowska, and G. Norman. Computing probability bounds for linear time formulas over concurrent probabilistic systems. In *PROBMIV’98, Electronics Notes in Theoretical Computer Science 22*, 1999.

[3] M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Towards a tool environment for model-based testing with AsmL. In *3rd Int. Workshop on Formal Approaches to Testing of Software, FATES’03, LNCS 2931*, pages 252–266. Springer, 2003.

[4] M. Bernardo. Non-bisimulation-based Markovian behavioral equivalences. *Journal of Logic and Algebraic Programming*, 72(1):3–49, 2007.

[5] M. Bernardo and W.R. Cleaveland. A theory of testing for markovian processes. In *11th Int. Conf. on Concurrency Theory, CONCUR’2000, LNCS 1877*, pages 305–319. Springer, 2000.

[6] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202(1-2):1–54, 1998.

[7] G. Bernot, L. Bouaziz, and P. Le Gall. A theory of probabilistic functional testing. In *19th IEEE Int. Conf. on Software Engineering, ICSE’97*, pages 216–226. ACM Press, 1997.

[8] L. Bottaci and E.S. Mresa. Efficiency of mutation operators and selective mutation strategies: An empirical study. *Software Testing, Verification and Reliability*, 9(4):205–232, 1999.

[9] M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-Markov processes. *Theoretical Computer Science*, 282(1):5–32, 2002.

[10] D.A. Carrington and P.A: Stocks. A tale of two paradigms: Formal methods and software testing. In *Z User Workshop*, pages 51–68. Springer, Workshops in Computing, 1994.

[11] 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.

[12] T. Y. Chen and Y. T. Yu. On the expected number of failures detected by subdomain testing and random testing. *IEEE Transactions on Software Engineering*, 4(2):109–119, 1996.

[13] T.S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.

[14] D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *3rd Workshop on Object-Oriented Real-Time Dependable Systems, WORDS'97*, pages 199–206. IEEE Computer Society Press, 1997.

[15] R. Cleaveland, Z. Dayar, S.A. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93–148, 1999.

[16] J.-M. Couvreur, N. Saheb, and G. Sutre. An optimal automata approach to ltl model checking of probabilistic systems. In *10th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR'03, LNCS 2850*, pages 361–375. Springer, 2003.

[17] R. de Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[18] M. Duflot, M.Z. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of bluetooth device discovery. *International Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.

[19] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-method: Testing real time systems. *IEEE Transactions on Software Engineering*, 28(11):1024–1039, 2002.

[20] S. Fujiwara, G. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite-state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991.

[21] A. Gill. *Introduction to The Theory of Finite State Machines*. McGraw–Hill, 1962.

[22] 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.

[23] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[24] R.M. Hierons. Testing from a non-deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.

[25] R.M. Hierons and M.G. Merayo. Mutation testing from probabilistic finite state machines. In *Third Workshop on Mutation Analysis, Mutation 2007*, pages 141–150. IEEE Computer Society Press, 2007.

[26] R.M. Hierons and H. Ural. Reduced length checking sequences. *IEEE Transactions on Computers*, 51(9):1111–1117, 2002.

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

[28] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[29] P. G. Hoel. *Elementary Statistics*. John Wiley and Sons, second edition, 1966.

[30] W.E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8:371–379, 1982.

[31] I. Hwang, T. Kim, S. Hong, and J. Lee. Test selection for a nondeterministic FSM. *Computer Communications*, 24(12):1213–1223, 2001.

[32] ITU-T. *Recommendation Z.500 Framework on formal methods in conformance testing*. International Telecommunications Union, Geneva, Switzerland, 1997.

[33] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal of Software Tools for Technology Transfer*, 6(2):128–142, 2004.

[34] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[35] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[36] Y. Lei, R.H. Carver, R. Kacker, and D. Kung. A combinatorial testing strategy for concurrent programs. *Software Testing, Verification and Reliability*, 17(4):207–225, 2007.

[37] N. López and M. Núñez. A testing theory for generally distributed stochastic processes. In *12th Int. Conf. on Concurrency Theory, CONCUR'01, LNCS 2154*, pages 321–335. Springer, 2001.

[38] G.L. Luo, G. von Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–161, 1994.

[39] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real time systems from logic specifications. *ACM Transactions on Computer Systems*, 13(4):356–398, 1995.

[40] M.G. Merayo and M. Núñez. Testing conformance on stochastic stream X-machines. In *5th Software Engineering and Formal Methods, SEFM'07*, pages 227–236. IEEE Computer Society Press, 2007.

[41] M.G. Merayo, M. Núñez, and I. Rodríguez. Testing finite state machines presenting stochastic time and timeouts. In *4th European Performance Engineering Workshop, EPEW'07, LNCS 4748*, pages 97–111. Springer, 2007.

[42] A. Nandi and R. Marculescu. System-level power/performance analysis for embedded systems design. In *38th ACM Design Automation Conference, DAC'01*.

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

[44] M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.

[45] M. Núñez and I. Rodríguez. Towards testing stochastic timed systems. In *23rd IFIP WG 6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems, FORTE'03, LNCS 2767*, pages 335–350. Springer, 2003.

[46] A. Petrenko, N. Yevtushenko, and G. von Bochmann. Testing deterministic implementations from their nondeterministic FSM specifications. In *9th IFIP Workshop on Testing of Communicating Systems, IWTCS'96*, pages 125–140. Chapman & Hall, 1996.

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

[48] I. Rouvellou and G. W. Hart. Inference of a probabilistic finite state machine from its output. *IEEE Transactions on Systems, Manufacturing and Cybernetics*, 25(3):424–437, 1995.

[49] J. C. Maldonado S. C. P. F. Fabbri, M. E. Delamaro and P. C. Masiero. Mutation analysis testing for finite state machines. In *5th IEEE International Symposium on Software Reliability Engineering, ISSRE 1994*, pages 220–229. IEEE Computer Society Press, 1994.

[50] P. C. Masiero M. E. Delamaro S. C. P. F. Fabbri, J. C. Maldonado and W. E. Wong. Mutation testing applied to validate specifications based on petri nets. In *8th International Conference on Formal Description Techniques, FORTE 95*, pages 329–337. Chapman & Hall, 1995.

[51] T. Sugeta S. C. P. F. Fabbri, J. C. Maldonado and P. C. Masiero. Mutation testing applied to validate specifications based on statecharts. In *10th IEEE International Symposium on Software Reliability Engineering, ISSRE 1999*, pages 210–219. IEEE Computer Society Press, 1999.

[52] R. Segala. Testing probabilistic automata. In *7th Int. Conf. on Concurrency Theory, CONCUR'96, LNCS 1119*, pages 299–314. Springer, 1996.

[53] K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *16th Int. Conf. on Computer Aided Verification, CAV'04, LNCS 3114*, pages 202–215. Springer, 2004.

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

[55] J. C. Maldonado T. Sugeta and W. E. Wong. Mutation testing applied to validate sdl specifications. In *16th IFIP International Conerence on Testing of Communicating Systems,TestCom 2004, LNCS 2978*, pages 193–208. Springer, 2004.

[56] W. Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21:216–227, 1992.

[57] M. Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *5th Int. AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems, ARTS'99, LNCS 1601*, pages 265–276. Springer, 1999.

[58] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. Carrasco. Probabilistic finite-state machines — part 1. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1013–1025, 2005.

[59] S.-H. Wu, S.A. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1-2):1–37, 1997.

[60] F. Zhang and T. Y. Cheung. Optimal transfer trees and distinguishing trees for testing observable nondeterministic finite-state machines. *IEEE Transactions on Software Engineering*, 29(1):1–14, 2003.