Testing from a Stochastic Timed System with a Fault Model [⋆]

Robert M. Hierons [a] Mercedes G. Merayo [a,b] Manuel Núñez [b]

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

[b] *Dep. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, 28040 Madrid, Spain*

Abstract

In this paper we present a method for testing a system against a non-deterministic stochastic finite state machine. As usual, we assume that the functional behaviour of the system under test (SUT) is deterministic but we allow the timing to be non-deterministic. We extend the state counting method of deriving tests, adapting it to the presence of temporal requirements represented by means of random variables. The notion of conformance is introduced using an implementation relation considering temporal aspects and the limitations imposed by a black-box framework. We propose an algorithm for generating a test suite that determines the conformance of a deterministic SUT with respect to a non-deterministic specification. We show how previous work on testing from stochastic systems can be encoded into the framework presented in this paper as an instantiation of our parameterized implementation relation. In this setting, we use a notion of conformance up to a given confidence level.

Key words: software testing, stochastic time, non-determinism, fault model.

1 Introduction

Formal analysis techniques rely on the idea of constructing a *formal model* that represents the critical aspects of the system under study. These models allow

[⋆] This research was carried out while the first author was visiting Universidad Complutense de Madrid under a grant *Programa de visitantes distinguidos e investigadores extranjeros en la UCM (Grupo Santander)*. Research partially supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01) and the Marie Curie project MRTN-CT-2003-505121/TAROT.

the developer to perform a systematic analysis that would be harder, or ever impossible, to apply to the system itself. The model can also be used to define the specification of a system being constructed. In order to use a formal technique, we need a model of the system under study expressed using a formal language. During the last two decades, the original formal languages have become more expressive. Thus, a new generation of languages have appeared to allow the explicit representation of non-functional aspects of systems (for example, the probability of performing a task [13,9,5,32,24,6] or the time consumed by the system while performing tasks, being either given by fixed amounts of time [38,31,14] or defined in probabilistic/stochastic terms [19,1,16,23,3]).

One of the advantages of using a formal approach to develop complex systems is that we can check the system's correctness with respect to the specification by comparing its empirical behaviour with that of the model. In this context, *formal testing techniques* [22,34,4,39] provide systematic procedures to check implementations in such a way that the coverage of critical parts/aspects of the system under test depends less on the intuition of the tester. In particular, they allow us to test the correctness of a system with respect to a specification. Formal testing originally targeted the functional behaviour of systems, such as determining whether the tested system can, on the one hand, perform certain actions and, on the other hand, does not perform some unexpected ones. While the relevant aspects of some systems only concern *what* they do, in some other systems it is equally relevant *how* they do what they do. Thus, after the initial consolidation stage, formal testing techniques started also to deal with non-functional properties. In particular, the work on formal testing applied to timed systems has attracted a lot of attention during the last years. In fact, there are already several proposals for timed testing (e.g. [26,8,42,41,10,11,21,2,30,29]). In these papers, time is usually considered to be *deterministic*, that is, time requirements follow the form "after/before t time units...".

A suitable representation of the temporal behaviour is critical for constructing useful models of real-time systems. In this paper we consider an extension of finite state machines (FSMs), *Stochastic Finite Sate Machines* (SFSMs), where (stochastic) time information is included. The duration of the actions are stochastically defined by means of *random variables*. That is, instead of having expressions such as "the action o takes t time units to be performed" we will have expressions such as "the time taken to perform action o will be drawn from distribution ξ." The use of *stochastic time* introduces several technical difficulties. For example, since the time that the system takes to perform an action may vary in different executions, even in the absence of (functional) non-determinism, the same sequence of actions may take different amounts of time in different runs of the system. In addition, we have to reformulate the notion of correctness; when testing from a SFSM we have to take into account not only the actions that the machine performs but also the temporal requirements. One possibility is to consider *equivalence*, that is, the system under test (SUT) is correct if and only if the sets of input/output sequences allowed by the SUT and the specification are identical and the delay associated with each sequence that can be performed by the SUT has a random variable with the same distribution as the corresponding random variable in the specification. An alternative notion of correctness, which is more appropriate when the specification is non-deterministic, relaxes this to allow the input/output sequences in the SUT to be a subset of those in the specification.

When testing from an FSM it is normal to make certain assumptions and a checking experiment is a test suite that is guaranteed to determine correctness under these assumptions. These assumptions are phrased in terms of the SUT behaving like an unknown element of a fault model. Most approaches for selecting a test suite from a non-deterministic FSM are based on *state counting* [43,37,36]. This paper considers the case where the specification may be a non-deterministic SFSM and extends the state counting based test selection method to the case of SFSMs. This is proven to provide full fault coverage on implementations whose functional behaviour is deterministic when using the standard assumption that we have a known upper bound on the number of states of a minimal SFSM that models the SUT.

Regarding temporal conformance, we have to take into account the fact that, in a black-box testing framework, we cannot access the random variables of the SUT. In fact, if we would consider equivalence of random variables, we would need an infinite number of observations from a random variable of the implementation (with an unknown distribution) to assure that this random variable is distributed as another random variable from the specification (with a known distribution). In addition, there are different notions of a random variable ξ' in the SUT following a probability distribution function F' conforming to a random variable ξ in the specification following a probability distribution function F. We give an implementation relation that is parameterized with a definition $\leq$ of what it means for a distribution in the SUT to conform to a distribution in the specification. For example, we may require the distributions to be equivalent or instead for them to have the same mean and thus allowing different choices for $\leq$ effectively makes SFSMs a more expressive formalism.

The main problem with the framework explained so far is that, by keeping such a high abstraction level, it makes it difficult to realize how the framework can be useful in black-box testing. In particular, by considering that implementation relations are parameterized by a certain relation among random variables, it might seem that we really need to have access to the random variables, specifically to their distributions, that appear in the implementation. In this line, the work presented in [33], and its subsequent adaptations to other frameworks [27,28], presents a more *realistic* approach. There, the idea is to check that for all appropriate input/output sequences, the execution time values observed while testing the SUT *fit* the random variables indicated by the specification. In testing terms we sample from the distribution ξ' in the SUT and use results from statistical sampling theory to compare the set of observations with ξ. However, the proposal introduced in [33] has a big drawback: It is not implementable in practice. This is because the test suite produced by the test derivation algorithm is, in general, infinite. Thus, in practice, we cannot emit a verdict regarding the correctness of the SUT. However, the ideas underlying [33] can be used to provide an interesting instantiation of the general method introduced in this paper. Essentially, by combining the notions of *sampling* and *fitting* with the adaption of state counting to a stochastic model, we will be able to define a new framework where we can provide a verdict, up to a certain confidence level, regarding implementations having a bound on its number of states.

This paper thus makes the following contributions. First, it defines a parameterizable implementation relation that allows us to capture a range of notions of conformance. Second, it gives an algorithm that, for a standard fault model and such an implementation relation, produces a test suite that allows us to determine whether the SUT conforms to the specification up to a required level of confidence. Finally, it shows that the notion of testing from stochastic systems presented in [33] can be *implemented* within the scope of the general method presented in this paper.

The rest of the paper is organized as follows. In the next section we introduce basic concepts and the notion of SFSM. In Section 3 we introduce a class of implementation relations that take into account both functional and timed aspects. In Section 4 we review the notions of reaching and distinguishing states that will be applied in the State Counting method and develop these for SFSMs. In Section 5 we establish the relationship between a construct called the Product Machine and the implementation relation defined in Section 3. In Section 6 we show how we can generate a finite test suite that can distinguish faulty implementations. In Section 7 we describe how previous work on testing from SFSMs can be appropriately represented in the general framework previously introduced. Finally, in Section 8 we present our conclusions and some lines of future work. In the appendix of the paper we formally define how hypothesis contrasts can be performed.

2 Preliminaries

In this section we introduce the main basic concepts that we will use in the paper. Specifically, we will give notation regarding finite automata as well as the languages recognized by them, we will introduce the notions of random variable and sample, and we will define the stochastic extension of the finite state machine model that we will use in this paper.

2.1 Alphabets and sequences

Throughout this paper, X denotes the set of inputs for the model and Y denotes the set of possible outputs for the model. Given a set A, A^* denotes the set of sequences composed of elements of A, including the empty sequence ϵ. In this paper, a variable that represents a sequence will have a bar over it, an example being $\bar{x}$.

2.2 Finite Automata

A finite automaton (FA) is defined by a tuple $F = (S, A, h, s_0, S_F)$ in which S is a finite set of states, A is the finite alphabet, h is the state transfer relation of type $S \times A \leftrightarrow S$, $s_0 \in S$ is the initial state and $S_F \subseteq S$ is the set of final states. If $s' \in h(s, a)$ then (s, a, s') is a *transition* and this means that if F receives a when in state s then it can move to state s'. F is a *deterministic FA (DFA)* if for all $s \in S$ and $a \in A$ there is at most one state $s' \in S$ such that $s' \in h(s, a)$.

The FA F defines a language $L(F)$: The set of sequences that can take F from its initial state to some final state in S_F. Thus, we have

$$L(F) = \{\bar{a} \in A^* | h(s_0, \bar{a}) \cap S_F \neq \emptyset\}$$

Similarly, the state s of F has the corresponding language, defined as the set

$$L_s(F) = \{\bar{a} \in A^* | h(s, \bar{a}) \cap S_F \neq \emptyset\}$$

Two FA or states are *equivalent* if they define the same language. It is known that every FA is equivalent to a DFA (see, for example, [12]).

2.3 Random variables and samples

We will denote by $\mathcal{V}$ the set of random variables. Let ξ be a random variable. We define its *probability distribution function* as the function $F_\xi : \mathbb{R} \longrightarrow [0, 1]$ 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. In order to avoid side-effects, we will always assume that all the random variables appearing in the definition of a system 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.

We will use *samples* to denote any multiset of positive real numbers. We denote the set of multisets in $\mathbb{R}^+$ by $\wp(\mathbb{R}^+)$. Let ξ be a random variable and Smp be a sample. We denote by $\gamma(\xi, Smp)$ the *confidence* of ξ on Smp. In our framework, samples will be associated with time values that implementations take to perform actions. We have that $\gamma(\xi, Smp)$ takes values in the interval $[0, 1]$. Intuitively, bigger values of $\gamma(\xi, Smp)$ indicate that the observed sample Smp is more likely to be produced by the random variable ξ. That is, this function decides how *similar* the probability distribution function generated by Smp and the one corresponding to the random variable ξ are. In the appendix of this paper we show how confidence is formally defined.

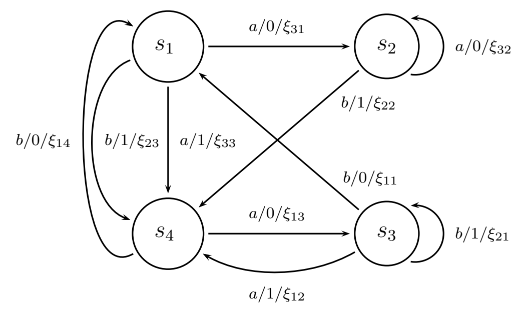
2.4 Stochastic Finite State Machines

A Stochastic Timed Finite State Machine can be seen as an FSM in which every transition also has a random variable that represents the expected distribution of times to execute the transition. Even though we use a slightly different notation to represent these machines, the underlying model coincides in fact with the one presented in [33].

Definition 1 *A Stochastic (Timed) Finite State Machine (SFSM) M is defined by a tuple (S, X, Y, δ, s_0) in which S is a finite set of states, X is the finite input alphabet, Y is the finite output alphabet, δ is the state transfer relation of type $S \times X \leftrightarrow Y \times \mathcal{V} \times S$, and $s_0 \in S$ is the initial state. If $(y, \xi, s') \in \delta(s, x)$ then (s, x, y, ξ, s') is a transition of M.*

Throughout this paper we assume that we are testing against an SFSM $M = (S, X, Y, \delta, s_0)$ with n states. A transition (s, x, y, ξ, s') should be interpreted in the following way: If M receives input x when in state s then it can take this transition in which case it outputs y, moves to state s', and introduces a delay t given by ξ.

Example 1 *Let us consider the machine depicted in Figure 1 in which the initial state is s_1. Each transition has an associated random variable. In the following we explain how these random variables are distributed. We consider that ξ_{1i}, for all $1 \leq i \leq 4$, are uniformly distributed in the interval $[0, 5]$. Uniform distributions assign equal probability to all the times in the interval. The random variables ξ_{2i}, for all $1 \leq i \leq 3$, follow a Dirac distribution in 4. The idea is that the corresponding delay will be equal to 4 time units. Finally, the random variables ξ_{3i}, for all $1 \leq i \leq 3$, are exponentially distributed with parameter 2. Let us consider the transition $(s_3, a, 1, \xi_{12}, s_4)$. Intuitively, if the machine is in state s_3 and receives the input a then it will produce the output*



$$F_1(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ \frac{x}{5} & \text{if } 0 < x < 5 \\ 1 & \text{if } x \geq 5 \end{cases}$$

$$F_2(x) = \begin{cases} 0 & \text{if } x < 4 \\ 1 & \text{if } x \geq 4 \end{cases}$$

$$F_3(x) = \begin{cases} 1 - e^{-2\cdot x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

where the random variables $\xi_{11}, \xi_{12}, \xi_{13}, \xi_{14}$ follow F_1, the random variables $\xi_{21}, \xi_{22}, \xi_{23}$ follow F_2, and the random variables $\xi_{31}, \xi_{32}, \xi_{33}$ follow F_3.

Fig. 1. Example of Stochastic Finite State Machine.

1 *after a time given by* ξ_{12}. *Since* ξ_{12} *follows a uniform distribution in the interval* $[0, 5]$*, we will have, for example, 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 transition has been performed (that is, we have probability* 1*).*

Any SFSM M can be represented by a FA $F(M) = (S, A, h, s_0, S)$ in which $A = X \times Y \times \mathcal{V}$ and $s' \in h(s, (x, y, \xi))$ if and only if $(y, \xi, s') \in \delta(s, x)$. The SFSM M is said to be *deterministic* if for all $s \in S$ and $x \in X$ we have that $|\delta(s, x)| \leq 1$. The SFSM M is *input-enabled* if for all $s \in S$ and $x \in X$, δ is defined on (s, x). In this paper we assume that any SFSM considered is input-enabled. Let us remark that this is not a real restriction on the expressivity of the model. In fact, any non input-enabled machine can be easily transformed into an input-enabled one just by adding a transition $(s, x, \mathit{null}, \xi, s)$ to each state s that originally did not have a transition labelled by x, where $\mathit{null} \notin Y$ is a new output and $\xi \in \mathcal{V}$ is any random variable. Thus, it is not difficult to relax the assumption that M is input-enabled: Essentially, in this paper we use input sequences that distinguish states of the SFSM and care would be required in defining what it means for an input sequence to distinguish two states.

We define two functions that are projections of δ. The function δ_1 models the state transfers and thus for all $s \in S$ and $x \in X$ we have that

$$\delta_1(s, x) = \{s' \in S | \exists y \in Y, \xi \in \mathcal{V}.(y, \xi, s') \in \delta(s, x)\}$$

Similarly, δ_2 models the outputs and delays and thus for all $s \in S$ and $x \in X$ we have that

$$\delta_2(s, x) = \{(y, \xi) | \exists s' \in S.(y, \xi, s') \in \delta(s, x)\}$$

We can extend these to take input sequences using the following recursive rules in which $x \in X$ and $\bar{x} \in X^*$:

- $\delta_1(s, \epsilon) = \{s\}$
- $\delta_1(s, x\bar{x}) = \{s' \in S | \exists s'' \in \delta_1(s, x).s' \in \delta_1(s'', \bar{x})\}$
- $\delta_2(s, \epsilon) = \{\epsilon\}$
- $\delta_2(s, x\bar{x}) = \{(y, \xi)\delta_2(s', \bar{x}) | (y, \xi, s') \in \delta(s, x)\}$

Since the specification SFSM can be non-deterministic, δ_1 and δ_2 can return sets that contain more than one element.

If we apply an input x when in state s of M and execute a transition that has output y then we use $\delta_y(s, x)$ to denote the set of possible next states. Thus, we have

$$\delta_y(s, x) = \{s' \in S | \exists \xi \in \mathcal{V}.(y, \xi, s') \in \delta(s, x)\}$$

Naturally we can extend these definitions in a similar manner to δ_1 and δ_2. Thus, for $x \in X$, $\bar{x} \in X^*$, $\bar{y} \in Y^*$, $y \in Y$ we have

- $\delta_\epsilon(s, \epsilon) = s$
- $\delta_{y\bar{y}}(s, x\bar{x}) = \delta_{\bar{y}}(\delta_y(s, x), \bar{x})$

If $\bar{x} = x_1 x_2 \ldots x_k$ and $\bar{y} = y_1 y_2 \ldots y_k$, with $x_i \in X$ and $y_j \in Y$, and $\delta_{\bar{y}}(s_0, \bar{x})$ is defined then we say that M can perform the sequence $x_1/y_1, x_2/y_2, \ldots, x_k/y_k$ and we say that this sequence is an *evolution* of M. We denote by $\texttt{NTEvol}(M)$ the set containing all the evolutions of M.

We can define another function, also taking as basis the different δ functions, to compute the random variable that we can obtain after performing a non-empty sequence. Formally,

- $\delta_y^{\mathcal{V}}(s, x) = \xi$, if $\exists s' \in S.\delta(s, x) = (y, \xi, s')$
- $\delta_{y\bar{y}}^{\mathcal{V}}(s, x\bar{x}) = \delta_{\bar{y}}^{\mathcal{V}}(\delta_y(s, x), \bar{x})$

Let us remark that this function will be undefined if either the sequences have different lengths or one of the transitions cannot be performed. If $\bar{x} = x_1 x_2 \ldots x_k$, $\bar{y} = y_1 y_2 \ldots y_k$, and $\bar{e} = x_1/y_1, x_2/y_2, \ldots, x_k/y_k$ then we will consider that $\delta^{\mathcal{V}}(s, \bar{e})$ denotes $\delta_{\bar{y}}^{\mathcal{V}}(s, \bar{x})$.

If for all $s \in S$, $x \in X$ and $y \in Y$ there do not exist $\xi_1 \neq \xi_2$ such that $(y, \xi_1) \in \delta_2(s, x)$ and $(y, \xi_2) \in \delta_2(s, x)$ then M is said to be *observable*. Let us note that observable machines allow non-determinism. For example, we can have a SFSM with two transitions such as (s, x, y, ξ, s_1) and (s, x, y', ξ', s_1') as long as $y \neq y'$. It is worth noticing that whereas an un-timed finite state machine is observable if and only if the associated FA is deterministic, in the case of SFSMs we only can claim that if an SFSM M is observable then $F(M)$ is deterministic. For example, we can consider a deterministic FA $F(M)$ that represents a SFSM M. $F(M)$ may contain the transitions $(s, (x, \xi_1, y), s_1)$ and

$(s, (x, \xi_2, y), s_2)$ which fulfill the deterministic condition of the FA but M is not observable. We only consider observable SFSMs in this paper. Thus, if $\delta_y(s, x) = \{s'\}$ then we write $\delta_y(s, x) = s'$.

3 Conformance and Fault Models

In order to reason about test effectiveness it is normal to introduce a fault model that contains a set of models with the property that it is believed that the SUT is equivalent to some (unknown) element of this set. The fault model used in this paper is defined in Section 3.1. In order to test against a specification M it is necessary to say what it means for the SUT to conform to M. The notion of conformance used is represented by an implementation relation. In Section 3.2 we describe the implementation relation used in this paper.

3.1 Fault models for SFSMs

A fault model Φ is a set of models with the property that the tester believes that the SUT is equivalent to an unknown element of Φ [20]. It is normal to describe the elements of the fault model using the formal language used to define the specification and thus the fault model will contain SFSMs. It is also usual to assume that we know the set of possible inputs and outputs of the SUT and in order to simplify the exposition we assume that these are the sets X and Y respectively. If we believe that the SUT could produce certain outputs that M cannot then we can extend Y. The implementation relation used in this paper ensures that we do not have to consider input values that are not in X. As usual in conformance testing, we assume that there is a given integer m such that it is known that the SUT has no more than m states (see, for example, [7,36]). These assumptions lead to the following fault model.

Definition 2 *Let $M = (S, X, Y, \delta, s_0)$ be a SFSM. The fault model Φ_M^m contains the set of deterministic input-enabled SFSMs that have input alphabet X, output alphabet Y, and at most m states.*

We thus assume that the SUT can be modelled by an unknown deterministic SFSM $M_I = (U, X, Y, \delta^I, u_0)$ that has at most m states. Let us recall that the condition that M_I is deterministic requires that the state transitions are deterministic but does not force the time delay for a transition to be a constant. It thus requires the functional behaviour of M_I to be deterministic but allows stochastic temporal behaviour.

Given the fact that the delay in a transition t of M is represented by a random variable ξ, it seems that a natural notion of conformance is to require the corresponding transition of the SUT to also have a delay represented by ξ. We then have the following definition of conformance.

Definition 3 *Let M be an SFSM and $M_I \in \Phi_M^m$. We say that M_I conforms to M, denoted by M_I conf M, if $L(F(M_I)) \subseteq L(F(M))$.*

Note that Definition 3 requires the transitions in the SUT to have identical distributions to the corresponding transitions in the specification.

In general, we cannot determine the random variable ξ' associated with a transition in the SUT through testing. However, we can assume that there is such a distribution ξ' in the SUT and thus that either the SUT M_I conforms to M or it does not. The problem then is to try to decide whether M_I conforms to M in black-box testing, where all we can do is to record the delays that the implementation needs to perform the transition and estimate ξ' on the basis of this. The approach described in this paper involves multiple executions of a transition, each execution leading to a time being recorded. The resultant set of times can then be compared with the expected distribution ξ using results from statistics. In this paper we use hypothesis contrasts, as presented in the Appendix, to decide up to a certain degree of confidence whether the observed times could be produced by the distribution governing the behavior of ξ. In testing we will therefore check that the distribution in the SUT conforms to the distribution in the specification up to a given confidence level, the tester choosing the required confidence level based on factors such as risk.

We have observed that Definition 3 requires the execution times of the transitions in the SUT to have the same distributions as the corresponding transitions in the specification. However, we can use weaker notions since timing requirements can be looser than this. For example, we might require that a random variable ξ' in the SUT has the same mean as the corresponding random variable ξ in the specification but to not insist that ξ' and ξ have the same distribution. As a consequence we might use a weaker notion of a random variable ξ' in M_I conforming to a random variable ξ in M and in order to make the results in this paper more widely applicable we simply denote this $\xi' \leq \xi$. Thus, the use of the $\leq$ relation between random variables does in fact constitute a general framework that can be instantiated, by giving a specific definition of $\leq$, when needed. We can now parameterize our definition of conformance with $\leq$.

Definition 4 *Let M be a SFSM and $M_I \in \Phi_M^m$. We say M_I $\leq$-conforms to M, denoted by M_I $conf_{\leq}$ M, if for all sequence $(x_1, y_1, \xi_1') \ldots (x_k, y_k, \xi_k') \in$*

$L(F(M_I))$ there exists a sequence $(x_1, y_1, \xi_1) \ldots (x_k, y_k, \xi_k) \in L(F(M))$ such that for all $1 \leq i \leq k$ we have $\xi'_k \leq \xi_k$.

We say that a state u of SFSM M_I $\leq$-conforms to a state s of SFSM M if for all sequence $(x_1, y_1, \xi'_1) \ldots (x_k, y_k, \xi'_k) \in L_u(F(M_I))$ there exists a sequence $(x_1, y_1, \xi_1) \ldots (x_k, y_k, \xi_k) \in L_s(F(M))$ such that for all $1 \leq i \leq k$ we have $\xi'_k \leq \xi_k$.

This new notion of conformance makes SFSMs more expressive since, by choosing an appropriate instance of this conformance relation, we can express properties such as "the mean transition execution time should be 1 second" while if we restrict ourselves to the conformance relation traditionally used with SFSMs then we cannot express such requirements.

We can also say what it means for a sequence in $(Y \times \mathcal{V})^*$ to $\leq$-conform to another sequence in $(Y \times \mathcal{V})^*$.

Definition 5 *Let $\bar{z} = (y_1, \xi_1) \ldots (y_k, \xi_k)$ and $\bar{z}' = (y'_1, \xi'_1) \ldots (y'_k, \xi'_k)$ be sequences in $(Y \times \mathcal{V})^*$. We say that $\bar{z}'$ $\leq$-conforms to $\bar{z}$ if for all $1 \leq i \leq k$ we have that $y_i = y'_i$ and $\xi'_i \leq \xi_i$. This is denoted $\bar{z}'\ conf_{\leq}\ \bar{z}$.*

4 Reaching and distinguishing states

This section discusses how we can produce input sequences to reach and distinguish states; these sequences will be used in the test generation algorithm given in Section 6. When testing from a deterministic FSM, it is normal to test a transition t by applying an input sequence that reaches the starting state of t, then the input from t and finally input sequences that distinguish between the ending state of t and all other states of the FSM (see, for example, [7,15,18,40]). However, for non-deterministic FSMs there need not exist an input sequence that is guaranteed to reach a particular state s. Further, there may be no input sequence that is guaranteed to distinguish between non-equivalent states s and s', since we may have that every input/output sequence that is possible from one of these states is also possible from the other state despite these states not being equivalent. Similar issues arise when testing from SFSMs, but are complicated by the presence of random variables governing the timed behaviour of transitions.

If there is an input sequence that is guaranteed to take SFSM M from s_0 to state s then s is said to be *deterministically-reachable*. This sequence allows us to reach a state of the SUT that should conform to the state s of M. Due to non-determinism in the specification there may be a state s of M such that there is no input sequence that is guaranteed to take M from s_0 to s. Other sequences allow us to distinguish between states of M. For example, the input sequence $\bar{x}$ allows us to distinguish between s and s' if no output sequence that can be produced by applying $\bar{x}$ in state s can also be produced by applying $\bar{x}$ in state s'. We can extend this to using sets of input sequences or adaptive test cases, leading to a more general concept, r-distinguishable states. Next, we formally define both of these concepts in the context of SFSMs.

Definition 6 *Let $M = (S, X, Y, \delta, s_0)$ be a SFSM and $s \in S$. We say that s is* deterministically-reachable *(*d-reachable*) if there is an input sequence $\bar{x}_s$ such that $\delta_1(s_0, \bar{x}_s) = \{s\}$ and then $\bar{x}_s$ is said to* deterministically-reach *(*d-reach*) s.*

Example 2 *The state s_4 in the machine represented in Figure 1 is d-reached by b while the state s_3 is d-reached by ba. On the contrary, the state s_2 is not d-reachable.*

It is also useful to be able to reason about the state of the SUT reached by a given sequence. It transpires that this is assisted by using input sequences that distinguish between states of M in a manner that guarantees that they distinguish between the corresponding states of the SFSM M_I that models the SUT if no failure is observed in testing. In order to distinguish between two states it is first worth considering how we can distinguish between two transitions $t_1 = (s_1, x, y_1, \xi_1, s'_1)$ and $t_2 = (s_2, x, y_2, \xi_2, s'_2)$ of SFSM M.

Definition 7 *Let $M = (S, X, Y, \delta, s_0)$ be a SFSM. Transitions $t_1 = (s_1, x, y_1, \xi_1, s'_1)$ and $t_2 = (s_2, x, y_2, \xi_2, s'_2)$ of M are* distinguishable *if either of the following hold:*

- *they produce a different output and so $y_1 \neq y_2$; or*
- *we can distinguish between ξ_1 and ξ_2 through it not being possible for a random variable $\xi' \in \mathcal{V}$ to conform to both ξ_1 and ξ_2, that is, for all $\xi' \in \mathcal{V}$ we have that $\neg(\xi' \leq \xi_1 \wedge \xi' \leq \xi_2)$.*

In this case we also say that (y_1, ξ_1) and (y_2, ξ_2) are distinguishable *and otherwise they are* compatible*. Similarly, if t_1 and t_2 are not distinguishable then we say that they are* compatible*.*

Two sequences $t_1 \dots t_k$ and $t'_1 \dots t'_k$ of transitions are compatible *if for all $1 \leq i \leq k$ we have that t_i and t'_i are compatible. Similarly, sequences $(y_1, \xi_1) \dots (y_k, \xi_k)$ and $(y'_1, \xi'_1) \dots (y'_k, \xi'_k)$ are compatible if for all $1 \leq i \leq k$ we have that (y_i, ξ_i) and (y'_i, ξ'_i) are compatible.*

We can now define what it means for an input sequence $\bar{x} \in X^*$ to distinguish between states s and s' of M and a natural way of doing this is to insist that we have no compatible sequences of transitions from s and s' with $\bar{x}$. This can be generalized by applying a similar approach as with un-timed finite state machines, inductively defining a set of input sequences (see, for

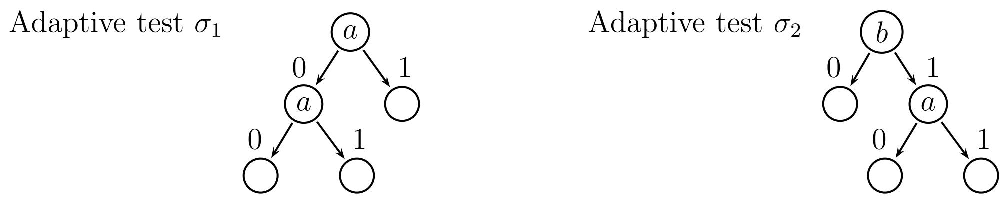


Fig. 2. Examples of Adaptive Tests.

example, [36]). An alternative is to use a set of finite adaptive test cases [17], where an adaptive test case σ is either the adaptive test case *null*, that involves no testing, or a pair (x, f) in which x is the next input to apply and f is a mapping from outputs to adaptive test cases. In the use of $\sigma = (x, f)$, first x is applied to the SUT, the resultant output y is observed and then the adaptive test case $f(y)$ is applied. Adaptive test cases thus correspond to finite decision trees. For example, the first tree in Figure 2 shows an adaptive test case in which the first input applied is a, represented by the contents of the 'top' node. If output 1 is observed in response to testing then the adaptive test case has ended, as represented by the arc with label 1 to an empty node. If instead 0 is observed then we next apply a and then the adaptive test case stops irrespective of the next output observed.

For a given input set X and output set Y, the set $\mathcal{T}(X,Y)$ of finite adaptive test cases can be recursively defined in the following way (see, for example [17]).

Definition 8 *Each element of $\mathcal{T}(X,Y)$ is either null or a pair (x, f) in which $x \in X$ and f is a mapping from Y to $\mathcal{T}(X,Y)$.*

In some situations it will be useful to define an adaptive test as a sequence of inputs followed by an adaptive test. The idea is that we do not explicitly consider the outputs that we receive while applying the initial input sequence because they do not influence our choices regarding future input. For example, if we have a state s being $d-$reached by an input sequence $\bar{x}$ and we are testing a transition from this state, we can apply first this sequence of inputs and then start the proper testing of the transition. In particular, we will use this notion in Section 6 when we produce test suites.

Definition 9 *Let $\sigma \in \mathcal{T}(X,Y)$ be an adaptive test case and $\bar{x}$ be a sequence of inputs. We recursively define the adaptive test case $\bar{x}\sigma$ as:*

$$\bar{x}\sigma = \begin{cases} \sigma & \text{if } \bar{x} = \epsilon \\ (x_1, f) & \text{if } \bar{x} = x_1\overline{x}_1 \ \wedge \forall y \in Y: \ f(y) = \overline{x}_1\sigma \end{cases}$$

If A is a set of input sequences and $\mathcal{T}$ is a set of adaptive test cases then we let $A\mathcal{T}$ denote the set of adaptive test cases formed by composing each input *sequence in A with each adaptive test case from $\mathcal{T}$.*

We can define what it means for an adaptive test case to r-distinguish two states. While from this it is straightforward to also produce a set of input sequences that r-distinguishes two states, in this paper we use adaptive test cases since they can lead to fewer sequences being applied in testing.

Definition 10 *Let $M = (S, X, Y, \delta, s_0)$ be a SFSM. We say that states s_1 and s_2 of M are* r(1)-distinguished *by the adaptive test case (x, f) if for every transition t_1 with start state s_1 and input x and every transition t_2 with start state s_2 and input x we have that t_1 and t_2 are distinguishable. We say that states s_1 and s_2 are* r(k)-distinguished *by the adaptive test case $\sigma = (x, f)$ $(k > 1)$ if either:*

- *there exists $1 \leq j < k$ such that s_1 and s_2 are r(j)-distinguished by σ; or*
- *for all $(y_1, \xi_1, s_1') \in \delta(s_1, x)$ and $(y_2, \xi_2, s_2') \in \delta(s_2, x)$ we have that either (y_1, ξ_1) and (y_2, ξ_2) are distinguishable or there exists $1 \leq j < k$ such that the states s_1' and s_2' are r(j)-distinguished by $f(y_1)$ $(y_1 = y_2)$.*

States s_1 and s_2 are r(k)-distinguishable if there exists an adaptive test case that r(k)-distinguishes them. States s_1 and s_2 are r-distinguishable if there exists $k \geq 1$ such that s_1 and s_2 are r(k)-distinguishable.

Throughout this paper W denotes a set of adaptive test cases that r-distinguishes all r-distinguishable pairs of states of M. If no two states of M are r-distinguishable then $W = \{null\}$.

Example 3 *Let us consider the FSM presented in Figure 1 and the adaptive test cases σ_1 and σ_2 given in Figure 2. The states s_1 and s_2 are not r-distinguishable since input a can take both states to s_2 with output 0 and two delays identically distributed (that is, ξ_{31} and ξ_{32}) while input b can take both states to s_4 with output 1 and two delays identically distributed (that is, ξ_{22} and ξ_{23}). Adaptive test case σ_1 r-distinguishes s_4 from all of the other states and r-distinguishes s_2 and s_3. Further, σ_2 r-distinguishes s_1 from s_3 and s_4 and so $W = \{\sigma_1, \sigma_2\}$ r-distinguishes all r-distinguishable states of the SFSM.*

5 Test effectiveness and the Product Machine

A number of methods for generating test suites from (un-timed) non-deterministic finite state machines have been based on the notion of a *Product Machine* (see, for example, [36,17,35]). In this section we extend the concept of a Product Machine to the situation in which we have stochastic delays.

The Product Machine can be thought of as the specification M and the (unknown) model M_I of the SUT running in parallel. Let us consider a state (s, u) of the Product Machine and a transition (u, x, y, ξ', u') of M_I. If M has a transition of the form (s, x, y, ξ, s') where $\xi' \leq \xi$ then the Product Machine contains the transition $((s, u), x, y, \xi', (s', u'))$; otherwise the Product Machine contains the transition $((s, u), x, y, \xi', Fail)$ for a special state $Fail$. The state $Fail$ thus represents the possibility of taking a transition in the SUT M_I for which there currently is no compatible transition in the specification M. Since M_I is unknown, the Product Machine is also unknown, however, it can be used in reasoning about test effectiveness.

Definition 11 *Let $M = (S, X, Y, \delta, s_0)$ and $M_I = (U, X, Y, \delta^I, u_0)$ be SFSMs. The Product Machine of M and M_I is the SFSM $P(M, M_I) = (P, X, Y, \delta^P, p_0)$ in which $p_0 = (s_0, u_0)$, $P = (S \times U) \cup \{Fail\}$, where $Fail \notin S \times U$, and δ^P is defined by the following:*

(1) If $(y, \xi', u') \in \delta^I(u, x)$ and there exists a state $s' \in S$ and ξ such that $(y, \xi, s') \in \delta(s, x)$ and $\xi' \leq \xi$ then $(y, \xi', (s', u')) \in \delta^P((s, u), x)$.

(2) If $(y, \xi', u') \in \delta^I(u, x)$ and there does not exist a state $s' \in S$ and ξ such that $(y, \xi, s') \in \delta(s, x)$ and $\xi' \leq \xi$ then $(y, \xi', Fail) \in \delta^P((s, u), x)$.

Note that the Product Machine need not be input-enabled since, in particular, no transitions are defined from the state $Fail$. An immediate consequence of the definition is that since M_I is deterministic, and so δ^I is a function, and M is observable the Product Machine is also deterministic.

Lemma 1 *Given SFSMs $M = (S, X, Y, \delta, s_0)$ and $M_I = (U, X, Y, \delta^I, u_0)$, if M is observable and δ^I is a function then δ^P is a function.*

Proof: Let us suppose that (s, u) is a state of the Product Machine and that $(y, \xi', (s', u')) \in \delta^P((s, u), x)$ for some $x \in X$. It is sufficient to prove that $\delta^P((s, u), x)$ contains no other elements. By definition, since δ^I is a function the state $Fail$ is not reachable from (s, u) using input x. Next we proceed with the proof by contradiction: Let us suppose that $(y_1, \xi'_1, (s'_1, u'_1)) \in \delta^P((s, u), x)$ for some $(y_1, \xi'_1, (s'_1, u'_1)) \neq (y, \xi', (s', u'))$. First, since δ^I is a function we have that $y_1 = y$, $\xi'_1 = \xi'$, and $u'_1 = u'$. Second, since M is observable and $y = y_1$ we must have that $s'_1 = s'$. This provides a contradiction as required. $\square$

Next we prove the announced result, that is, the SUT conforms to M if and only if the state $Fail$ of the Product Machine is not reachable.

Theorem 1 *Let $M = (S, X, Y, \delta, s_0)$ be an observable and input-enabled SFSM and $M_I = (U, X, Y, \delta^I, u_0)$ be in Φ_M^m. M_I conf$_\leq$ M if and only if the state Fail of $P(M, M_I)$ is unreachable.*

Proof: First let us assume that the state $Fail$ of the Product Machine is reachable. We will use proof by contraction. Thus, let us suppose that M_I

does $\leq$-conform to M. Let $\bar{x}$ denote a minimum length input sequence that can take $P(M, M_I)$ from p_0 to the state $Fail$ and let $\bar{x} = \bar{x}_1 x$ for $x \in X$, $\bar{x}_1 \in X^*$. Thus $\bar{x}_1$ reaches a state (s, u) of the Product Machine such that there exist $y \in Y$ and $\xi' \in \mathcal{V}$ such that $(y, \xi') = \delta_2^I(u, x)$ and there is no $(y, \xi) \in \delta_2(s, x)$ such that $\xi' \leq \xi$. Let $\bar{z}' = \delta_2^I(u_0, \bar{x}_1)$ and let $\bar{z}$ denote the unique element of $\delta_2(s_0, \bar{x}_1)$ such that $\bar{z}'$ $conf_{\leq}$ $\bar{z}$: uniqueness is ensured since M is observable. By the uniqueness of $\bar{z}$, since M_I $conf_{\leq}$ M and so $\bar{z}'(y, \xi')$ $\leq$-conforms to an element of $\delta_2(s_0, \bar{x})$, $\delta_2(s_0, \bar{x})$ must contain a sequence in the form $\bar{z}(y, \xi)$ for some ξ such that $\xi' \leq \xi$. Since M is observable, $\delta_{\bar{z}}(s_0, \bar{x}_1) = s$ and so $(y, \xi) \in \delta_2(s, x)$ for some ξ with $\xi' \leq \xi$. This provides a contradiction as required.

Now let us assume that the state $Fail$ of M is unreachable. Again, we will perform the proof by contraction by supposing that M_I does not $\leq$-conform to M. Let $\bar{x}$ denote a minimum length input sequence such that $\delta_2^I(u_0, \bar{x})$ does not $\leq$-conform to any element of $\delta_2(s_0, \bar{x})$ and let $\bar{x} = \bar{x}_1 x$, for $x \in X$ and $\bar{x}_1 \in X^*$. Let us also consider $\bar{z}' = \delta_2^I(u_0, \bar{x}_1)$. By the minimality of $\bar{x}$, there is some $\bar{z} \in \delta_2(s_0, \bar{x}_1)$ such that $\bar{z}' \leq \bar{z}$ and so $\delta_1^P(p_0, \bar{x}_1) = (s, u)$ for some $(s, u) \in S \times U$. Thus, since $\delta_2^I(u_0, \bar{x})$ does not $\leq$-conform to any element of $\delta_2(s_0, \bar{x})$ we have that $\delta_2^I(u, x)$ does not $\leq$-conform to any element of $\delta_2(s, x)$. By the definition of δ^P, $Fail = \delta_1^P((s, u), x)$ and so, since (s, u) is reachable, the state $Fail$ is reachable. This provides a contradiction as required. $\square$

Testing can thus be seen as a process of executing the SUT in order to determine whether the state $Fail$ of the unknown Product Machine is reachable. In Section 6 we show how state counting can be adapted in order to produce a test suite $\mathcal{T}$ with the property that the state $Fail$ of the Product Machine is reachable if and only if it is reached by some element of $\mathcal{T}$. This allows us to test in order to determine whether the SUT conforms to M with a given confidence.

The following result says that an input sequence $\bar{x}$ reaches the state $Fail$ of the Product Machine if and only if it triggers a sequence of transitions in the SUT with a label $\bar{z}'$ such that there is no corresponding sequence of transitions in M: If we *observe* $\bar{z}'$ in testing then we have observed a failure.

Proposition 1 *Let $M = (S, X, Y, \delta, s_0)$ and $M = (U, X, Y, \delta^I, u_0)$ be in Φ_M^m. If an input sequence $\bar{x}$ reaches state Fail of the Product Machine $P(M, M_I)$ then it triggers a sequence of transitions in M_I with a label $\bar{z}'$ such that there is no sequence $\bar{z} \in L(F(M))$ such that $\bar{z}'$ $conf_{\leq}$ $\bar{z}$.*

Proof: Proof by induction on the length of $\bar{x}$. Since $Fail$ is not reachable by the empty sequence, the result holds for the base case ϵ. The inductive hypothesis is that the result holds for all input sequences of length less than k and let $\bar{x}$ be an input sequence of length k that reaches the state $Fail$. Let

$\bar{x} = \bar{x}_1 x$ for $x \in X$. If $\bar{x}_1$ reaches the state $Fail$ then the result follows from the inductive hypothesis. We therefore assume that $\bar{x}_1$ reaches a state (s, u) of the Product Machine.

Let $\bar{z}'_1 = \delta^I_2(u_0, \bar{x}_1)$ and let $\bar{z}_1$ be the unique element of $\delta_2(s_0, \bar{x}_1)$ such that $\bar{z}'_1$ $conf_{\leq}$ $\bar{z}_1$. There exists $y \in Y$ and $\xi' \in \mathcal{V}$ such that $(y, \xi') = \delta^I_2(u, x)$ and so $\bar{z}'_1(y, \xi') = \delta^I_2(u_0, \bar{x})$.

Since $\bar{x}$ reaches the state $Fail$, there is no $(y, \xi) \in \delta_2(s, x)$ such that $\xi' \leq \xi$. Thus, by the uniqueness of $\bar{z}_1$, there does not exist $\bar{z} \in L(F(M))$ such that $\bar{z}'$ $conf_{\leq}$ $\bar{z}$ as required. $\square$

The following result shows how we can use a set W of adaptive test cases, as described in Section 4, in order to explore certain aspects of the structure of the SUT.

Proposition 2 *Let $M = (S, X, Y, \delta, s_0)$ and $M = (U, X, Y, \delta^I, u_0)$ be in Φ^m_M. Let us suppose that $\bar{x}_1$ and $\bar{x}_2$ reach states (s_1, u_1) and (s_2, u_2) respectively of the Product Machine $P(M, M_I)$. If the adaptive test case $\sigma = (x, f)$ r-distinguishes s_1 and s_2 and both $\bar{x}_1\sigma$ and $\bar{x}_2\sigma$ do not reach the state Fail of the Product Machine then $u_1 \neq u_2$.*

Proof: By definition, the result holds if and only if it holds for adaptive test cases that $r(k)$-distinguish states, for all $k \geq 1$. The proof will proceed by induction on k. The base case, with $k = 1$, holds immediately.

Let us consider the inductive hypothesis that the result holds for all $k < l$, for some $l > 1$. It is sufficient to prove that it holds for $k = l$ and we therefore assume that σ $r(l)$-distinguishes s_1 and s_2. If s_1 and s_2 are $r(j)$-distinguishable, for some $j < l$, then the result follows immediately from the inductive hypothesis. If $\delta^I_2(u_1, x) \neq \delta^I_2(u_2, x)$ then the result holds so we assume that $\delta^I_2(u_1, x) = \delta^I_2(u_2, x) = (y, \xi')$ for some y and ξ'. Let $u'_i = \delta^I_1(u_0, \bar{x}_i x) = \delta^I_1(u_i, x)$ $(1 \leq i \leq 2)$. Since the state $Fail$ is neither reached by $\bar{x}_1\sigma$ nor by $\bar{x}_2\sigma$, and s_1 and s_2 are not $r(1)$-distinguishable, there exist some ξ and $s'_1, s'_2 \in S$ such that $(y, \xi, s'_i) = \delta(s_i, x)$ and $\xi' \leq \xi$ $(i \in \{1, 2\})$. Thus, we conclude $\delta^P((s_i, u_i), x) = (y, \xi', (s'_i, u'_i))$, for $i \in \{1, 2\}$. By definition, since σ $r(l)$-distinguishes s_1 and s_2, there exists $1 \leq j < l$ such that the states s'_1 and s'_2 are $r(j)$-distinguished by $f(y)$ and so, from the inductive hypothesis, we derive that $u'_1 \neq u'_2$. The result thus follows from M_I being deterministic. $\square$

6 Test suite generation

We have seen that testing can be represented as a process of trying to determine whether the state $Fail$ of the (unknown) Product Machine is reachable. In this section we assume that we are testing against an SFSM M with m states using the fault model Φ_M^m. Thus the Product Machine has at most $mn + 1$ states and so the state $Fail$ of the Product Machine is reachable if and only if it can be reached using an input sequence of length at most mn. We could thus use the test suite X^{mn} formed by applying Definition 9 to the composition of the input sequences of length mn and the adaptive test $null$. This section shows how we can use a smaller test suite if there are states that are d-reachable and/or states that are r-distinguishable.

The test suite will be developed using a breadth-first search starting from the d-reachable states of M. We need a termination criterion for this search and the criterion is based on the idea of searching for a *minimal* sequence that reaches $Fail$. Let us consider a d-reachable state s of M and a sequence $\bar{x}_s$ used to d-reach s. The termination criterion is satisfied by an input sequence $\bar{x}$ following $\bar{x}_s$ if we can show that $\bar{x}$ cannot be a prefix of a minimum length input sequence that reaches the state $Fail$ when followed by an element of W. The reasoning used to achieve this is based on an approach called *state counting*. We place a lower bound on the number of states that M_I must have if the input sequence, followed by any element of W, does not reach $Fail$ and no state of the Product Machine has been repeated (and so the input sequence can be the prefix of a minimal sequence that, when followed by an element of W, reaches $Fail$). We stop when this exceeds the upper bound on the number of states of M_I. We now introduce notation that will be used in state counting.

Let $\hat{S}$ denote the set of d-reachable states of M. Further, for each $s \in \hat{S}$ we let $\bar{x}_s$ denote an input sequence that d-reaches s and we fix $\bar{x}_{s_0} = \epsilon$. For each state $s \in \hat{S}$ let $\bar{z}'_s$ denote the sequence $(x_1, y_1, \xi'_1) \dots (x_k, y_k, \xi'_k)$ in $L(F(M_I))$ where $\bar{x}_s = x_1 \dots x_k$. We let $V = \{\bar{x}_s | s \in \hat{S}\}$ be the set of $\bar{x}_s$ for $s \in \hat{S}$; this set contains exactly one input sequence for each d-reachable state of M including the empty sequence ϵ. Given a set $S' \subseteq S$ of states, $\hat{S}' = S' \cap \hat{S}$ denotes the d-reachable states belonging to S'.

We assume that W is the set of adaptive test cases used to pairwise distinguish states of M. We let $S_1, \dots, S_k$ denote the maximal sets of states that are pairwise distinguished by W. We can now adapt the notion of state counting [36].

Let us recall that given SFSM M, $F(M)$ is the corresponding FA and for a state s of M, $L_s(F(M))$ denotes the language of sequences from $X \times Y \times \mathcal{V}$ defined by the paths in M from s. Given $\bar{z} = (x_1, y_1, \xi_1) \dots (x_r, y_r, \xi_r) \in L_s(F(M))$ we say that $\bar{z}$ *visits* the states $\delta_{y_1}(s, x_1), \delta_{y_1 y_2}(s, x_1 x_2), \dots, \delta_{y_1 \dots y_r}(s, x_1 \dots x_r)$.

Given a d-reachable state $s \in \hat{S}$, a set $Tr(s)$ (called a *traversal set* in [36]) can be defined:

- On the basis of the successor tree, generate the set $F_s \subseteq L_s(F(M))$ of sequences from $(X \times Y \times \mathcal{V})^*$ defined by: F_s is the set of $\bar{z} \in L_s(F(M))$ such that there exists $1 \leq i \leq k$ where $\bar{z}$ visits states from S_i exactly $m - |\hat{S}_i| + 1$ times from s and this condition does not hold for any proper prefix of $\bar{z}$.
- $Tr(s)$ is the set of input sequences such that there is some corresponding sequence in F_s: $Tr(s) = \{x_1 \dots x_r \in X^* | \exists y_1, \dots, y_r \in Y, \xi_1, \dots, \xi_r \in \mathcal{V}. (x_1, y_1, \xi_1) \dots (x_r, y_r, \xi_r) \in F_s\}$.

The test suite is formed in the following way: For each $s \in \hat{S}$ and sequence $\bar{x} \in Tr(s)$ we include each element of $\{\bar{x}_s\} pre(\bar{x}) W$ where $pre(\bar{x})$ is the set of prefixes of $\bar{x}$, including the empty sequence ϵ. Given state $s \in \hat{S}$ we let $T(s) = \{\bar{x}_s\} \cup_{\bar{x} \in Tr(s)} pre(\bar{x})$. Then we get the following test suite:

$$\mathcal{T} = \bigcup_{s \in \hat{S}} T(s) W$$

Lemma 2 *Let $M = (S, X, Y, \delta, s_0)$ be an observable input-enabled SFSM, $M_I = (U, X, Y, \delta^I, u_0)$ be in Φ_M^m. If no adaptive test case in $\mathcal{T}$ reaches the state $Fail$ of the Product Machine $P(M, M_I)$ then the state $Fail$ of $P(M, M_I)$ is not reachable.*

Proof: We will prove the result by contradiction. Thus, let us suppose that no adaptive test case in $\mathcal{T}$ reaches the state $Fail$ of the Product Machine $P(M, M_I)$ but that the state $Fail$ of $P(M, M_I)$ is reachable. Let us consider $s \in \hat{S}$ and $\bar{x} \in X^*$ such that $Fail = \delta_1^P((s_0, u_0), \bar{x}_s \bar{x})$ and $\bar{x}$ is minimal. In this situation there does not exist $s' \in \hat{S}$ and $\bar{x}' \in X^*$ such that $Fail = \delta_1^P((s_0, u_0), \bar{x}_{s'} \bar{x}')$ and $\bar{x}'$ is shorter than $\bar{x}$. Since $\epsilon \in V$ there exists some such s and $\bar{x}$. Let $\bar{x} = x_1 \dots x_l$ and let $\bar{z}' = z'_1 \dots z'_l = \delta_2^I(\delta_1^I(u_0, \bar{x}_s), \bar{x})$.

Since no adaptive test case in $\mathcal{T}$ reaches the state $Fail$ of the Product Machine $P(M, M_I)$, we must have that there is no $\bar{x}_1 \in Tr(s)$ such that $\bar{x} \in pre(\bar{x}_1)$. Let $\bar{x}_1 = x_1 \dots x_{k'}$ denote the longest prefix of $\bar{x}$ in $Tr(s)$ and let $\bar{z}'_1 = z'_1 \dots z'_{k'}$ denote the corresponding prefix of $\bar{z}'$. By the definition of $Tr(s)$, there exists $\bar{z}_1 = z_1 \dots z_{k'} \in (Y \times \mathcal{V})^*$, with $z_i = (y_i, \xi_i)$, such that $\bar{z}'_1 \; conf_{\leq} \; \bar{z}_1$, $(x_1, y_1, \xi_1) \dots (x_{k'}, y_{k'}, \xi_{k'})$ is contained in F_s and the inclusion of this in F_s is based on some set S_r.

For each $s_i \in S_r$ let $n_i = \{\bar{x}_s x_1 \dots x_j | 1 \leq j \leq k' \wedge s_i = \delta_{y_1 \dots y_j}(s, x_1 \dots x_j)\}$ be the non-empty prefixes of $\bar{x}_1$ that visit s_i from state s in M. By the minimality of $\bar{x}$ no state of the Product Machine is repeated and so the states of M_I reached by sequences in n_i must be distinct. Further, if $s_i \in \hat{S}_r$ then the state of M_I reached by $\bar{x}_{s_i}$ must not be reached by a sequence contained in n_i. Let

n'_i be defined by: If $s_i \in \hat{S}_r$ then $n'_i = n_i \cup \{\bar{x}_{s_i}\}$; otherwise $n'_i = n_i$. Then the states of M_I reached by sequences in n'_i must be distinct.

Further, since W pairwise distinguishes the states in S_r, by Proposition 2, for all $\bar{x}_i \in n'_i$ and $\bar{x}_j \in n'_j$, for states $s_i, s_j \in S_r$ with $s_i \neq s_j$, we have that $\delta^I(u_0, \bar{x}_i) \neq \delta^I(u_0, \bar{x}_j)$. Thus, M_I must have at least $\sum_{s_i \in S_r} |n'_i|$ states. Therefore, by the definition of F_s, M_I must have at least $m - |\hat{S}_r| + 1 + |\hat{S}_r| = m + 1$ states. Since M_I has at most m states, this provides a contradiction as required. □

Theorem 2 *If M is an observable input-enabled SFSM and $M_I \in \Phi_M^m$ then we have that no sequence in $\mathcal{T}$ reaches the state Fail of the Product Machine $P(M, M_I)$ if and only if M_I conf$_\leq$ M.*

Proof: First observe that by Lemma 2, no sequence in $\mathcal{T}$ reaches the state $Fail$ of the Product Machine $P(M, M_I)$ if and only if the state $Fail$ of $P(M, M_I)$ is not reachable. The result thus follows from Theorem 1. □

Example 4 *Let us consider the machine depicted in Figure 1. Previously we showed that all the states except s_2 are d-reachable. We choose a deterministic state cover $V = \{\epsilon, b, ba\}$. The states s_1 and s_2 are not r-distinguishable and the set of adaptive tests $W = \{\sigma_1, \sigma_2\}$ r-distinguishes the rest of states. We have two maximal sets of pairwise r-distinguishable states $S_1 = \{s_1, s_3, s_4\}$ and $S_2 = \{s_2, s_3, s_4\}$. The states in S_1 are d-reachable therefore $|\hat{S_1}| = 3$. Nevertheless the state s_2 in S_2 is not d-reachable and so $|\hat{S_2}| = 2$.*

We assume that the implementations has no more than 4 states ($m = n = 4$). The termination criterion for expanding the input sequences is reached if either states of $\hat{S_1}$ are visited twice ($m - |\hat{S_1}| + 1$) or states of $\hat{S_2}$ are reached three times ($m - |\hat{S_2}| + 1$) during the derivation of the sequence from each of the d-reachable states.

Figure 3 represents the successor tree corresponding to the d-reachable state s_4. All the paths to the leaves contain two nodes that represent states of S_1 except one of them that transverses three times the state s_2 of the set S_2. This tree leads to the transversal set

$$Tr(s_4) = \{\epsilon, a, b, aa, ab, ba, bb, baa, bab, baaa, baab\}$$

and culminates in

$$\mathcal{T}(s_4) = \{b\}Tr(s_4)W$$

We can therefore find a finite set $\mathcal{T}$ of input sequences such that the SUT conforms to M if and only if no input sequence in $\mathcal{T}$ reaches the state $Fail$ of the Product Machine. By Proposition 1, if an input sequence $\bar{x}$ reaches state

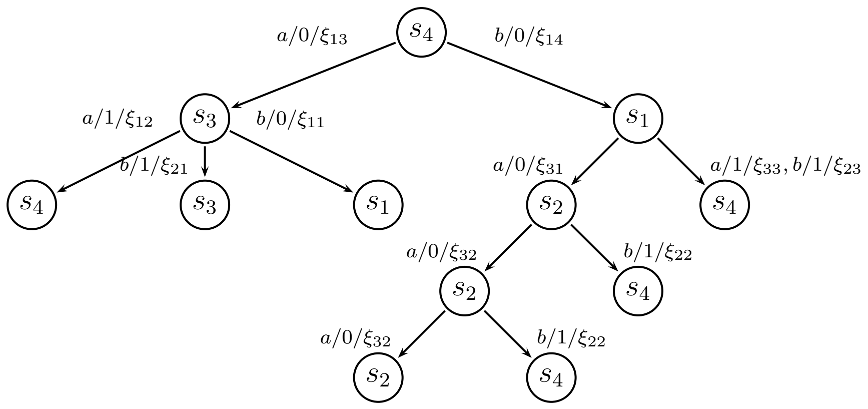


Fig. 3. Tree representing F_{s_4}.

Fail of the Product Machine then it triggers a sequence of transitions in the SUT with a label $\bar{z}'$ such that there is no sequence $\bar{z} \in L(F(M))$ such that $\bar{z}'$ *conf* $_{\leq}\bar{z}$. We therefore have to test to check this property for every $\bar{x} \in \mathcal{T}$ and in the following section we explain how hypothesis contrasts can be used. The most important benefit of the result in Theorem 2 is that the definition of conformance is in terms of an infinite set of input sequences and we have shown that it is sufficient to consider a finite set $\mathcal{T}$.

7 Implementing other notions of conformance

In this section we show how the testing framework presented in [33] can be adequately encoded into the framework described in this paper. However, these two approaches are not exactly the same and we have to deal with the differences in order to make a proper adaption. There are two main differences between them, in both cases, related to conformance regarding stochastic time. On the one hand, while conformance is established in [33] by considering the random variable obtained by adding all the random variables taking part in a sequence, in this paper we consider random variables for each transition of the sequence. On the other hand, the main notion of conformance given in [33] does not really compare random variables of the specification and the implementation, but it compares random variables of the specification with observed time values from testing the implementation.

We will do the encoding in two steps. First, we will consider an approximation that deals only with the first of the differences and then we will show how the previous study can be modified to take into account the second difference.

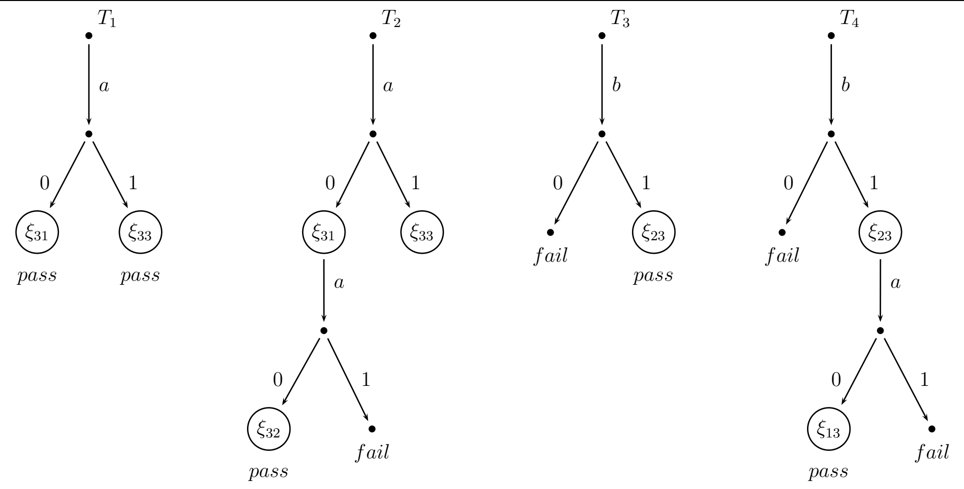


Fig. 4. Examples of Tests: $X = \{a, b\}$ and $Y = \{0, 1\}$.

7.1 A new notion of test

There are two main differences between tests as introduced in [33] and adaptive test cases as considered in this paper. First, in [33], after applying an input, the testing process can continue only after the reception of one specific output while in the current framework adaptive test cases can continue after one of several outputs. Second, in [33], once the testing process finishes we can return two verdicts, *pass* and *fail*, while adaptive test cases do not explicitly return a verdict. A schematic representation of tests can be seen in Figure 4. In addition, we have to modify the notion of test given in [33] to deal with the framework developed in this paper. Specifically, time will be observed not only when the testing process successfully finishes but also in all the intermediate phases.

Definition 12 *A test is a tuple $T = (S, X, Y, \delta, s_{in}, S_I, S_O, S_F, S_P, C_T)$ where S is the set of states, X and Y are disjoint sets of input and output actions, respectively, $\delta \subseteq S \times (X \cup Y) \times S$ is the transition relation, $s_{in} \in S$ is the initial state, and the sets $S_I, S_O, S_F, S_P \subseteq S$ are a partition of S. The transition relation and the sets of states fulfill the following conditions:*

- *S_I is the set of input states. We have that $s_{in} \in S_I$. For all input state $s \in S_I$ there exists a unique outgoing transition $(s, x, s') \in \delta$. For this transition we have that $x \in X$ and $s' \in S_O$.*
- *S_O is the set of output states. For all output state $s \in S_O$ we have that for all $y \in Y$ there exists a unique state s' such that $(s, y, s') \in \delta$. In this case, $s' \notin S_O$. Moreover, there do not exist $x \in X$ and $s' \in S$ such that $(s, x, s') \in \delta$.*
- *S_F and S_P are the sets of fail and pass states, respectively. We say that these states are terminal. That is, for all state $s \in S_F \cup S_P$ there do not*

exist $z \in X \cup Y$ and $s' \in S$ such that $(s, z, s') \in \delta$.

Finally, $C_T : (S_P \cup S_I - \{s_{in}\}) \longrightarrow \mathcal{V}$ is a function associating random variables with pass and input states.

Let $\bar{e} = x_1/y_1, \ldots, x_r/y_r$ be an input/output sequence and $s^T \in S$. We write $T \stackrel{\bar{e}}{\Longrightarrow} s^T$ if there exist states $s_{12}, s_{21}, s_{22}, \ldots s_{r1}, s_{r2} \in S$ such that for all $2 \leq j \leq r$ we have $(s_{j1}, x_j, s_{j2}) \in \delta$, for all $1 \leq j \leq r-1$ we have $(s_{j2}, y_j, s_{(j+1)1}) \in \delta$, and $\{(s_{in}, x_1, s_{12}), (s_{r2}, y_r, s^T)\} \subseteq \delta$.

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

If we are testing an implementation with input and output sets X and Y, respectively, tests are deterministic acyclic $X \cup Y$ labelled transition systems with a strict alternation between input actions and output actions. After an output action we may find either a leaf or another input action. Leaves can be labelled either by *pass* or by *fail*. In addition, we add random variables to both pass and input states. Let us remark that we do not consider a random variable in the initial state of the test because, at that point, the testing process has not started yet.

From now on we will assume that when we talk about tests we refer only to valid tests. Next we define the application of a test to an implementation. Let us recall that $\texttt{NTEvol}(M_I)$ is the set of input/output sequences from paths that start at the initial state of M_I.

Definition 13 *Let $T = (S', X, Y, \delta', s_{in}, S_I, S_O, S_F, S_P, C_T)$ be a valid test and $M_I = (S, X, Y, \delta, s_0)$ be a deterministic input-enabled SFSM. We denote the application of the test T to M_I by $M_I \parallel T$. Let $s^T \in S'$ be a state of T. We write $M_I \parallel T \stackrel{\bar{e}}{\Longrightarrow} s^T$ if $T \stackrel{\bar{e}}{\Longrightarrow} s^T$ and $\bar{e} \in \texttt{NTEvol}(M_I)$.*

We say that M_I passes T, denoted by $\texttt{pass}(M_I, T)$, if for all $\bar{e} \in \texttt{NTEvol}(M_I)$ we have that $M_I \parallel T \stackrel{\bar{e}}{\Longrightarrow} s^T$ implies s^T does not belong to S_F. We say that M_I stochastically passes T, denoted by $\texttt{pass}_{sto}(M_I, T)$, if $\texttt{pass}(M_I, T)$ and $M_I \parallel T \stackrel{\bar{e}}{\Longrightarrow} s^T$ implies $\delta^{\mathcal{V}}(s_{in}, \bar{e}) \leq C_T(s^T)$.

Let $\mathcal{T}S$ be a test suite. We say that M_I passes $\mathcal{T}S$, denoted by $\texttt{pass}(M_I, \mathcal{T}S)$, if for all $T \in \mathcal{T}S$ we have $\texttt{pass}(M_I, T)$. We say that M_I stochastically passes $\mathcal{T}S$, denoted by $\texttt{pass}_{sto}(M_I, \mathcal{T}S)$, if for all $T \in \mathcal{T}S$ we have $\texttt{pass}_{sto}(M_I, T)$.

Since we are assuming that implementations are input-enabled, the testing process will conclude only when the test reaches either a fail or a pass state. Moreover, if we are applying a test consisting of k inputs, in order to check whether a test is successfully passed we only need to consider those evolutions of M_I having at most k inputs. Finally, let us remark that in the definition of passing a test the states s^T cannot be output states since they are reached after performing an input/output sequence. Thus, either s^T is a fail state or $C_T(s^T)$ is defined.

We conclude this section by showing the relation between adaptive test cases, as introduced in Definition 8, and the alternative notion of test given in Definition 12. On the one hand, an adaptive test case can be used to bring the SUT to a point where it will perform a transition forbidden by the specification. On the other hand, the tests introduced in this section will return fail if they detect an erroneous, with respect to the specification, behaviour. Next we will show that, due to the nature of adaptive test cases, each adaptive test case will be represented by a *test suite*, the adaptive test case and the test suite having the same discriminatory power as shown in the forthcoming Proposition 3.

Definition 14 *Let $(x, f) \in \mathcal{T}(X, Y)$ be a non-empty adaptive test case and M be a SFSM. We define the test suite generated by (x, f) with respect to M, denoted by $\texttt{Generate}((x, f), M)$, as the set of tests that can be produced by applying the algorithm given in Figure 5. Given a set $\mathcal{T}S$ of adaptive test cases we let $\texttt{Generate}(\mathcal{T}S, M)$ denote the test suite generated by the elements of $\mathcal{T}S$ with respect to M and so $\texttt{Generate}(\mathcal{T}S, M) = \bigcup_{T \in \mathcal{T}S} \texttt{Generate}(T, M)$.*

Let us remark that a single application of the algorithm given in Figure 5 returns one test. However, by taking into consideration the different alternatives of the algorithm, iterative applications will produce the desired test suite.

The algorithm works as follows. It keeps a set of auxiliary states of the test that have not been completed yet. Each of these states can be considered a pass state (step (1) of the algorithm). In addition, at most one of these states can be used to continue the testing process (step (2) of the algorithm). In this case, we add a branch labelled by the input of the auxiliary adaptive test case and consider all the possible outputs. If the sequence performed from the initial state (that is, e_{aux} followed by the last input/output pair) cannot be performed by the specification then the considered output reaches a fail state. If this sequence can be performed but the adaptive test case concludes (that is, after the output we get *null*) then the considered output reaches a pass state. In the remaining case, the reached state is added to the set of auxiliary states and the rest of auxiliary variables are updated accordingly. In addition, in the last two cases, the random variable associated with the reached state (this state will be either a pass or an input state) is set to a random variable extracted from the specification.

Example 5 *In Figure 4, $\{T_1, T_2\}$ is the test suite generated by applying Definition 14 to the adaptive test case σ_1 given in Figure 2 with respect to the SFSM given in Figure 1. Similarly, $\{T_3, T_4\}$ is the test suite generated from σ_2.*

```
Input: A non-empty adaptive test case $(x, f)$ and a SFSM $M = (S, X, Y, \delta, s_0)$.
Output: A test $T = (S', X, Y, \delta', s_{in}, S_I, S_O, S_F, S_P, C_T)$.

$S' := \{s_{in}\}$;  $\delta', S_I, S_O, S_F, S_P := \emptyset$; $S_{aux} := \{(s_{in}, \epsilon, (x, f))\}$.
While $S_{aux} \neq \emptyset$ do
• Choose $(s^T, e_{aux}, (x_{aux}, f_{aux})) \in S_{aux}$.
• Choose one of the following two alternatives:
  {Second alternative can be chosen only if $S_{aux}$ is singleton}
(1) If $e_{aux} \neq \epsilon$ then {$s^T$ will be a pass state}
    (a) $S_{aux} := S_{aux} - \{(s^T, e_{aux}, (x_{aux}, f_{aux}))\}$;  $S_P := S_P \cup \{s^T\}$.
(2) If $S_{aux} = \{(s^T, e_{aux}, (x_{aux}, f_{aux}))\}$ then
    (a) $S_{aux} := \emptyset$.
    (b) Consider a fresh state $s' \notin S'$; $S' := S' \cup \{s'\}$.
    (c) $S_I := S_I \cup \{s^T\}$;  $S_O := S_O \cup \{s'\}$;  $\delta' := \delta' \cup \{(s^T, x_{aux}, s')\}$.
        {Add an input transition labelled by $x_{aux}$; then, consider
        all outputs}
    (d) For all $y \in Y$ do
        (i) Consider a fresh state $s_y \notin S'$; $S' := S' \cup \{s_y\}$.
        (ii) $\delta' := \delta' \cup \{(s', y, s_y)\}$.
        (iii) If $e_{aux} \cdot x_{aux}/y \notin$ NTEvol$(M)$ then $S_F := S_F \cup \{s_y\}$.
              {These outputs lead to a fail state since they are
              not expected by the specification}
        (iv) If $e_{aux} \cdot x_{aux}/y \in$ NTEvol$(M)$ and $f_{aux}(y) = null$ then
             $S_P := S_P \cup \{s_y\}$; $C_T(s_y) := \delta^{\mathcal{V}}(s_0, e_{aux} \cdot x_{aux}/y)$.
             {These outputs lead to a pass state since they are
             expected but the adaptive test case finished here}
        (v) If $e_{aux} \cdot x_{aux}/y \in$ NTEvol$(M)$ and $f_{aux}(y) \neq null$ then
            {These outputs are expected. In subsequent
            traversals of the loop, at most one of them will
            lead to an input state where the test continues; the
            rest will lead to pass states}
              · $C_T(s_y) := \delta^{\mathcal{V}}(s_0, e_{aux} \cdot x_{aux}/y)$.
              · $S_{aux} := S_{aux} \cup \{(s_y, e_{aux} \cdot x_{aux}/y, f_{aux}(y))\}$.
                {New auxiliary adaptive test case, $f_{aux}(y)$, is
                continuation of the previous one after $y$}
```

Fig. 5. Generation of tests from an adaptive test case.

The proof of the following is easy but cumbersome. It only consists in showing that an adaptive test case reaches *Fail* in the Product Machine if and only if the test suite is not passed. The proof follows from the fact that we have generated tests by taking into account the paths of the adaptive test case that are allowed by the specification. Thus, *null* adaptive test cases are correctly replaced by either pass or fail states.

Proposition 3 *Let M be a SFSM, $M_I \in \Phi_M^m$, and (x, f) be a non-empty* *adaptive test case. We have that no sequence in the adaptive test case* (x, f) *reaches the state* $Fail$ *of the Product Machine* $P(M, M_I)$ *if and only if we have* $\textbf{\textit{pass}}_{sto}(M_I, \textbf{\textit{Generate}}((x, f), M))$.

7.2 Test derivation in the new framework

```
Input: A SFSM $M = (S, X, Y, \delta, s_0)$ having $n$ states.
Output: A test $T = (S', X, Y, \delta', s_{in}, S_I, S_O, S_F, S_P, C_T)$.

$S' := \{s_{in}\};\ \delta', S_I, S_O, S_F, S_P := \emptyset;\ len := 0;\ S_{aux} := \{(s_0, s_{in})\}$.
While $S_{aux} \neq \emptyset$ do
• Choose $(s^M, s^T) \in S_{aux}$.
• Choose one of the following two alternatives:
(1) $S_{aux} := S_{aux} - \{(s^M, s^T)\};\ S_P := S_P \cup \{s^T\}$. {$s^T$ will be a pass state}
(2) If $S_{aux} = \{(s^M, s^T)\}$ and $len < nm$ then
    (a) $S_{aux} := \emptyset$; Choose $x \in X$.
    (b) Consider a fresh state $s' \notin S'$; $S' := S' \cup \{s'\}$.
    (c) $S_I := S_I \cup \{s^T\};\ S_O := S_O \cup \{s'\};\ \delta' := \delta' \cup \{(s^T, x, s')\};\ len := len+1$.
        {Add an input transition labelled by $x$; then, consider
        all the outputs}
    (d) For all $y \in Y$ such that $\nexists \xi \in \mathcal{V}.(y, \xi) \in \delta_2(s^M, x)$ do
        {These outputs lead to a fail state since they are not
        expected by the specification}
            · Consider a fresh state $s_y \notin S'$; $S' := S' \cup \{s_y\}$.
            · $S_F := S_F \cup \{s_y\};\ \delta' := \delta' \cup \{(s', y, s_y)\}$.
    (e) For all $y \in Y$ such that $\exists \xi \in \mathcal{V}.(y, \xi) \in \delta_2(s^M, x)$ do
        {These outputs are expected. At most one of them will
        lead to an input state where the test continues; the
        rest will lead to pass states}
            · Consider a fresh state $s_y \notin S'$; $S' := S' \cup \{s_y\}$.
            · $\delta' := \delta' \cup \{(s', y, s_y)\};\ C_T(s_y) := \xi$.
            · $s_1^M := \delta_y(s^M, x);\ S_{aux} := S_{aux} \cup \{(s_1^M, s_y)\}$.
```

Fig. 6. Derivation of tests from a specification.

As explained in Section 6, if we are checking the conformance of an implementation belonging to Φ_M^m, that is, a deterministic, input-enabled machine having at most m states, with respect to the specification M having n states, then we know that it is enough to consider all the tests having at most nm inputs. In the new framework we can also notably reduce this set by considering only *relevant* behaviours of the specification. Our derivation algorithm, given in Figure 6, is an adaption of the one given in [33]. The basic idea consists in traversing the specification in order to get all the possible evolutions in an appropriate way. By considering the possible available choices we get a test suite extracted from M. We denote this test suite by $\texttt{tests}(M)$. Next we explain how our algorithm works. A set of *pending situations* S_{aux} keeps track of those states of the test whose outgoing transitions have not been completed yet. More precisely, a pair $(s^M, s^T) \in S_{aux}$ indicates that we did not complete the state s^T of the test and the current state in the traversal of the specification is s^M. The set S_{aux} initially contains only a pair with the initial states of both SFSM and test. For each pair belonging to S_{aux} we may choose one possibility. It is important to remark that the second step can be applied only when the set S_{aux} becomes singleton. So, our derived tests correspond to valid tests as introduced in Definition 12. The first possibility simply indicates that the state of the test becomes a pass state. The second possibility takes an input and generates a transition in the test labelled by this input. This possibility can be taken only if the test is not *too long*, since we know that it is enough to restrict tests to have a number of inputs less than or equal to nm. Then, the whole sets of outputs is considered. If the output is not expected by the SFSM (step 2.(d) of the algorithm) then a transition leading to a fail state is created. This could be simulated by a single branch in the test, labelled by `else`, leading to a fail state (in the algorithm we suppose that *all* the possible outputs appear in the test). For the expected outputs (step 2.(e) of the algorithm) we create a transition with the corresponding output, add the appropriate pair to the set S_{aux}, and add the random variable labelling the corresponding transition of the specification.

Let us note that finite tests are constructed simply by considering a step where the second inductive case is not applied.

We now show that the test suites derived from specifications identify faults in a SUT if and only if the SUT does not conform to the specification. The proof of the result is based on the original proof [33] and taking into account Proposition 3.

Theorem 3 *Let M be a SFSM and $M_I \in \Phi_M^m$. We have M_I $\leq$-conforms to M if and only if $\texttt{pass}_{sto}(M_I, \texttt{tests}(M))$.*

If we put together the previous result and Theorem 2 we obtain the following result that shows the relation between the test derivation algorithms given in Sections 6 and 7.

Corollary 1 *Let M be a SFSM and $M_I \in \Phi_M^m$. We have that no sequence in the test suite $\mathcal{T} = \bigcup_{s \in \hat{S}} T(s)W$ introduced in Section 6, as returned by state counting, reaches the state $Fail$ of the Product Machine $P(M, M_I)$ if and only if $\texttt{pass}_{sto}(M_I, \texttt{tests}(M))$.*

Theorem 4 *Let M be a SFSM, $M_I \in \Phi_M^m$, and let $\mathcal{T} = \bigcup_{s \in \hat{S}} T(s)W$ be the test suite introduced in Section 6, as returned by state counting. We have M_I*

$\leq$-conforms to M if and only if $\mathtt{pass}_{sto}(M_I, \mathtt{Generate}(\mathcal{T}, M))$.

Proof: This follows from Theorem 2 and Proposition 3. $\square$

It is straightforward to show that $\mathtt{Generate}(\mathcal{T}, M) \subseteq \mathtt{tests}(M)$ and thus that we make testing more efficient by basing it on the test suite produced using state counting.

7.3 Another testing framework: Recording observed time values

As we commented before, there are two important differences regarding how testing is applied in [33] and in this paper. The first one, to consider random variables in all the intermediate states of the testing process, was treated in the previous section. In this section we show how the previous development has to be modified to deal with the second difference: To compare random variables of the specification with observed time values from the SUT.

When we test the SUT with respect to a SFSM we need to check not only that the emitted output after we apply each input of the test is the same as that specified. Systems with temporal requirements expressed by means of random variables require us to also check that time values that the SUT takes for producing each output fit with the random variable associated with the corresponding transition in the specification. In order to do this, we will collect a sample of time values and *compare* this sample with the random variable. By comparison we mean that we will apply a criterion to decide, with a certain confidence α, whether the sample could be generated by the corresponding random variable. We only need to assume that there exists a way to observe the time that the SUT takes to perform each step in testing. This will allow us to register the observed time executions obtained from the interaction with the implementation. Then, we will check that each sample matches the corresponding random variable up to an established confidence level. The notion of *matching* corresponds to the application of a hypothesis contrast to decide whether the sample could be generated by the corresponding random variable. In the appendix of this paper we show how such hypothesis contrast can be formally performed. In the following, given a sequence $\bar{t} = t_1, \ldots, t_l$ and $1 \leq j \leq l$ we let $\pi_k(\bar{t}) = t_k$ denote the k-th element of $\bar{t}$.

Definition 15 *Let M_I be a SFSM. We say that $x_1/y_1/t_1, \ldots, x_l/y_l/t_l$ is an observed timed execution of M_I if the observation of M_I shows that for all $1 \leq j \leq l$, the time elapsed between the acceptance of the input x_j and the observation of the output y_j is t_j time units. We will sometimes refer to $x_1/y_1/t_1, \ldots, x_l/y_l/t_l$ as $(\bar{e}, \bar{t})$ where $\bar{e} = x_1/y_1, \ldots, x_l/y_l$ and $\bar{t} = t_1, \ldots, t_l$. Finally, let $1 \leq k \leq l$. We denote by $preseq_k(\bar{e})$ the prefix of $\bar{e}$ having length* *k, that is,* $x_1/y_1, \ldots, x_k/y_k$.

Let $J = \{\!|(\bar{e}'_1, \bar{t}_1), \ldots, (\bar{e}'_p, \bar{t}_p)|\!\}$ *be a multiset of timed executions and* $\Upsilon = \{\bar{e}_1, \ldots, \bar{e}_m\}$ *be a set of input/output sequences. We say that the function* $\mathtt{Sampling}^k_{(J,\Upsilon)} : \Upsilon \longrightarrow \wp(I\!R^+)$ *is a k-sampling application of J for* Υ *if for all* $\bar{e} \in \Upsilon$ *we have*

$$\mathtt{Sampling}^k_{(J,\Upsilon)}(\bar{e}) = \{\!|\pi_k(\bar{t}) \mid \exists \bar{e}'.(\bar{e}', \bar{t}) \in J \wedge |\bar{e}'| \geq k \wedge preseq_k(\bar{e}) = preseq_k(\bar{e}')|\!\}$$

Timed executions are input/output sequences together with the time that it took to perform each transition. Regarding the definition of sampling applications, we just associate each proper prefix of an input/output sequence with the observed execution time values of the last step of the subsequence.

Definition 16 *Let M be a SFSM,* $M_I \in \Phi^m_M$*, J be a multiset of timed executions, and* $0 \leq \alpha \leq 1$*. Let us consider the set* $\Upsilon = \{\bar{e} \mid \exists\, \bar{t}.(\bar{e}, \bar{t}) \in J\}$*. We say that* M_I (α, J) $\leq$*-conforms to M if for all* $\bar{e} = x_1/y_1, \ldots, x_l/y_l \in \Upsilon$ *there exists some* $(x_1, y_1, \xi_1) \ldots (x_k, y_l, \xi_l) \in L(F(S))$ *such that for all* $1 \leq k \leq l$ *we have that* $\gamma(\xi_k, \mathtt{Sampling}^k_{(J,\Upsilon)}(\bar{e})) > \alpha$.

Intuitively, the observed time values corresponding to each prefix of the considered input/output sequences $\bar{e}$ match the definition of the corresponding random variable, that is, for all $1 \leq k \leq |\bar{e}|$ we have $\gamma(\xi_k, \mathtt{Sampling}^k_{(J,\Upsilon)}(\bar{e})) > \alpha$.

This implementation relation cannot be compared with the previous one since the methodologies are completely different: On the one hand we compare random variables while, on the other hand, we compare random variables and time values. Next, we have to modify the notion of passing a test given in Definition 13. The idea consists of applying time conditions to the set of *observed timed executions*, not to timed evolutions of the implementations. We need a set of test executions, associated with each evolution, to determine whether they match the probability distribution function associated with these random variables. In order to increase the degree of reliability, we will put together all the observations so that we have more instances for each evolution. In particular, some observations will be used several times. In other words, an observation from a given test may be used to check the validity of another test sharing the same observed sequence.

Definition 17 *Let* M_I *be a deterministic, input-enabled SFSM, T be a test, and* s^T *be a state of T. We write* $M_I \parallel T \overset{\bar{e}}{\Longrightarrow}_{\bar{t}} s^T$ *if* $T \overset{\bar{e}}{\Longrightarrow} s^T$ *and* $(\bar{e}, \bar{t})$ *is an observed timed execution of* M_I*. In this case we say that* $(\bar{e}, \bar{t})$ *is a test execution of* M_I *and T. We say that a set of test executions of* M_I *and T is a test execution sample of* M_I *and T.*

Let $\mathcal{T}_{st} = \{T_1, \ldots, T_n\}$ *be a test suite and* $J_1, \ldots, J_n$ *be test execution samples* *of M_I with $T_1, \ldots, T_n$, respectively. Let $J = \bigcup_{i=1}^{n} J_i$, $\Upsilon = \{\bar{e} \mid \exists\, \bar{t}.(\bar{e}, \bar{t}) \in J\}$ and let us consider $0 \leq \alpha \leq 1$. We say that M_I (α, J)- passes the test suite $\mathcal{T}_{st}$ if $\texttt{pass}(M_I, \mathcal{T}_{st})$ and for all $\bar{e} \in \Upsilon$ and all $T \in \mathcal{T}_{st}$ such that $M_I \parallel T \overset{\bar{e}}{\Longrightarrow} s^T$, we have that $\gamma(C_T(s^T), \texttt{Sampling}^{|\bar{e}|}_{(J,\Upsilon)}(\bar{e})) > \alpha$.*

Intuitively, a SUT passes a test if there does not exist an evolution leading to a fail state. Once we know that the functional behaviour of the implementation is correct with respect to the test, we need to check time conditions. The set J corresponds to the observations of the (several) applications to M_I of the tests belonging to the test suite $\mathcal{T}_{st}$. Let us intuitively explain the process. We will apply each test belonging to the test suite to the implementation several times. If we find an unexpected output then we stop the testing process and conclude that the implementation is faulty. If we do not find such an error, for each test we collect several observed timed executions corresponding to each time that the application of the test reached a pass state. Thus, we obtain for each test T_i a multiset $\{\!\!\{(\bar{e}^i_1, \bar{t}_{1i}), (\bar{e}^i_2, \bar{t}_{2i}), \ldots, (\bar{e}^i_m, \bar{t}_{mi})\}\!\!\}$. These multisets, more exactly the time values corresponding to each different evolution, will be used to make the hypothesis contrast. Thus, we have to decide whether, for each evolution $\bar{e}$, the observed time values corresponding to the last step of the evolution (that is, $\texttt{Sampling}^{|\bar{e}|}_{(J,\Upsilon)}(\bar{e})$) *match* the definition of the random variables appearing in the state of the test corresponding to the execution of that evolution (that is, $C_T(s^T)$).

The proof of the following result is a simple adaption of a similar result in [33].

Theorem 5 *Let M be a SFSM, $M_I \in \Phi^m_M$, J be a multiset of timed executions, and $0 \leq \alpha \leq 1$. We have that M_I (α, J) $\leq$-conforms to M if and only if M_I (α, J)- passes the test suite $\texttt{tests}(M)$.*

8 Conclusions

In this paper we have extended the state counting method of deriving tests from a non-deterministic finite state machine (FSM) to the case of non-deterministic stochastic FSMs. This model allows us to easily introduce time requirements for the performance of actions by associating random variables with the transitions. The notion of conformance has been represented by means of an implementation relation where functional and temporal conditions are considered, taking into account the restrictions imposed by a black-box framework. The timing aspects of the definition of conformance used is parameterizable, allowing the techniques developed in this paper to be applied using a range of implementation relations.

We have proposed a test generation algorithm, based on the presence of d-

reachable and r-distinguishable states in the specification and the notion of the Product Machine. This algorithm allows us to obtain a test suite $\mathcal{T}$ that determines the conformance of a deterministic SUT with respect to a non-deterministic specification with a given confidence. The required level of confidence can be chosen by the tester and testing then involved repeatedly applying each element of $\mathcal{T}$ a sufficient number of times, the number of times being determined by standard results from statistical sampling theory.

Finally, we have shown how previous relevant work on testing from stochastic systems can be considered as a particular case of the work developed in this paper. We have introduced a new notion of test to capture the alternative testing framework. In this line, we have proved that an adaptive test case has the same discriminatory power as a certain set of tests derived from the specification.

There are a number of areas of future work. While many implementations are deterministic, some are nondeterministic and so it would be interesting to extend the results to the testing of a non-deterministic SUT. In testing from an FSM it has been shown that sets of prefixed of sequences from a characterising set W can be used in state identification, leading to the Wp-method [25], and it seems likely that such an approach could be extended to SFSMs. The test effort depends both on the size of the test suite and the number of times each adaptive test case must be applied in order to provide the required level of confidence. One approach to optimisation is to produce a minimal test suite and then apply each adaptive test case a minimum number of times and this is essentially what we do in this paper. However, the number of times we have to apply an adaptive test case depends both on the required confidence and the distributions associated with the adaptive test case. It thus seems likely that better results will be provided by considering both aspects in one optimisation phase. Finally, we have not considered the problem of finding a set W that minimises the test effort.

Appendix - Statistics background: Hypothesis contrasts.

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

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

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

Definition 18 *Let ξ be a random variable and J be a multiset of real numbers representing a sample. Let X^2 be the discrepancy level of J on ξ calculated as explained above by splitting the sampling space into k classes*

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

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

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

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

Example 6 *Let us consider a device that produces real numbers belonging to the interval* $[0,1]$. *We would like to test whether the device produces these numbers randomly, that is, it does not have a number or sets of numbers that have a higher probability of being produced than others. Thus, we obtain a sample from the machine and we apply the contrast hypothesis to determine whether the machine follows a uniform distribution in the interval* $[0,1]$. *First, we have to decide how many classes we will use. Let us suppose that we take* $k = 10$ *classes. Thus, for all* $1 \leq i \leq 9$ *we have* $a_i = 0.i$ *and* $P(\xi \leq a_i) = \frac{i}{10}$. *So,* $C = \{[0,0.1),[0.1,0.2)\ldots[0.8,0.9),[0.9,\infty)\}$.

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

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

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

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

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

References

[1] 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.

[2] L. Brandán Briones and E. Brinksma. Testing real-time multi input-output systems. In *7th Int. Conf. on Formal Engineering Methods, ICFEM'05, LNCS 3785*, pages 264–279. Springer, 2005.

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

[4] E. Brinksma and J. Tretmans. Testing transition systems: An annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 187–195. Springer, 2001.

[5] 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.

[6] L. Cheung, M. Stoelinga, and F. Vaandrager. A testing scenario for probabilistic processes. *Journal of the ACM*, 54(6):Article 29, 2007.

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

[8] 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.

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

[10] 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.

[11] M.A. Fecko, M.Ü. Uyar, A.Y. Duale, and P.D. Amer. A technique to generate feasible tests for communications systems with multiple timers. *IEEE/ACM Transactions on Networking*, 11(5):796–809, 2003.

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

[13] 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.

[14] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.

[15] F.C. Hennie. Fault-detecting experiments for sequential circuits. In *5th Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, 1964.

[16] H. Hermanns. *Interactive Markov Chains.* PhD thesis, Universität Erlangen-Nürnberg, 1998. Also appeared as *LNCS 2428*, Springer, 2002.

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

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

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

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

[21] K.G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using Uppaal. In *4th Int. Workshop on Formal Approaches to Testing of Software, FATES'04, LNCS 3395*, pages 79–94. Springer, 2004.

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

[23] 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.

[24] N. López, M. Núñez, and I. Rodríguez. Specification, testing and implementation relations for symbolic-probabilistic systems. *Theoretical Computer Science*, 353(1–3):228–248, 2006.

[25] 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.

[26] 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.

[27] M.G. Merayo, M. Núñez, and I. Rodríguez. Implementation relations for stochastic finite state machines. In *3rd European Performance Engineering Workshop, EPEW'06, LNCS 3964*, pages 123–137. Springer, 2006.

[28] 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.

[29] M.G. Merayo, M. Núñez, and I. Rodríguez. Extending EFSMs to specify and test timed systems with action durations and timeouts. *IEEE Transactions on Computers*, 57(6):835–848, 2008.

[30] M.G. Merayo, M. Núñez, and I. Rodríguez. Formal testing from timed finite state machines. *Computer Networks*, 52(2):432–460, 2008.

[31] 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.

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

[33] 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.

[34] A. Petrenko. Fault model-driven test derivation from finite state models: Annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 196–205. Springer, 2001.

[35] A. Petrenko and N. Yevtushenko. Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, 54(9):1154–1165, 2005.

[36] 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.

[37] A. Petrenko, N. Yevtushenko, A.V. Lebedev, and A. Das. Nondeterministic state machines in protocol conformance testing. In *6th IFIP Workshop on Protocol Test Systems, IWPTS'93*, pages 363–378. North Holland, 1993.

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

[39] I. Rodríguez, M.G. Merayo, and M. Núñez. $\mathcal{HOTL}$: Hypotheses and observations testing logic. *Journal of Logic and Algebraic Programming*, 74(2):57–93, 2008.

[40] D.P. Sidhu and T.-K. Leung. Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering*, 15(4):413–426, 1989.

[41] 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.

[42] M.Ü. Uyar, M.A. Fecko, A.S. Sethi, and P.D. Amar. Testing protocols modeled as FSMs with timing parameters. *Computer Networks*, 31(18):1967–1998, 1999.

[43] N. Yevtushenko, A.V. Lebedev, and A. Petrenko. On checking experiments with nondeterministic automata. *Automatic Control and Computer Sciences*, 6:81–85, 1991.