Model-Based Testing of Stochastic Systems with IOCO Theory

Marcus Gerhold
Formal Methods and Tools
University of Twente, The Netherlands
m.gerhold@utwente.nl

Mariëlle Stoelinga
Formal Methods and Tools
University of Twente, The Netherlands
marielle@cs.utwente.nl

ABSTRACT

We present essential concepts of a model-based testing framework for probabilistic systems with continuous time. Markov automata are used as an underlying model. Key result of the work is the solid core of a probabilistic test theory, that incorporates real-time stochastic behaviour. We connect **ioco** theory and hypothesis testing to infer about trace probabilities. We show that our conformance relation conservatively extends **ioco** and discuss the meaning of quiescence in the presence of exponentially distributed time delays.

CCS Concepts

•**General and reference** → **Validation;** •**Software and its engineering** → **Formal methods;** *Empirical software validation;* •**Theory of computation** → *Program specifications;*

Keywords

Model-Based Testing, Markov automata, **ioco**

1. INTRODUCTION

Probability plays an increasingly important role in many applications. A large body of randomized algorithms, protocols and computation methods use randomization to achieve their goals. Security protocols use random bits in their encryption methods [8]; routing in sensor networks facilitates random walks [1] and control policies in robotics assign bandwidth in a random fashion. Moreover, the explosive growth of embedded software causes an increasing need of sophisticated modelling methods. Reactive systems demand techniques to assess quantitative aspects used to model speed or delay.

A key question is then how to effectively test the correctness of such systems. One possible answer are verification techniques. Verification of probabilistic processes matured to a major research field, presenting models like probabilistic automata [34], Markov decision processes [31] and generalized stochastic Petri nets [27] with techniques like stochastic model checking [32] and tools like Prism [22] or Plasma [20].

In practice, testing is the most common validation technique. The system under test (SUT) is subjected to a large body of well-designed test cases and the outcome of the process is compared to a requirements specification. Model-based testing (MBT) emerged as a means to automate this process. It gained popularity in industry by providing efficient methods to generate, execute and evaluate test cases from a specification pinning down the desired behaviour.

A wide variety of MBT frameworks exist, capable of handling different properties of a system, such as functional properties [37], real-time properties [6, 23] and quantitative aspects [4]. Notably, a vast majority of the frameworks are based on Tretmans' input output conformance relation christened **ioco** [38], or the closely related *may/must* relation [30]. However, frameworks for probabilistic systems with continuous time are underdeveloped.

Our approach. Our requirements specifications are given as Markov automata (MA). MAs incorporate both stochastic time and discrete probability. They form the semantic foundation of dynamic fault and defence tree analysis [33], generalised stochastic Petri nets [28] and the standardised modeling language AADL [5].

Mathematically MAs arise as the conservative extension of both probabilistic automata (PAs) [34] and interactive Markov chains (IMCs) [17]. MAs are equipped with both probabilistic and nondeterministic choices. The first represent the choices made by the system (e.g. coin tosses) or the environment (e.g. degradation rates, failure probabilities). The latter model choices, that are not under its control. As argued in [34] these are important for implementation freedom, scheduler choices and interleaving. Complementary, they are of particular interest because of their memoryless exponential distributions inherited from IMCs. These give a highly appropriate stochastic approximation, if only the mean duration of an activity is known.

The challenge of our work consists of combining discrete probability choices made the SUT, exponentially distributed time delays and nondeterministic choices. We use schedulers (a.k.a. adversaries) to resolve nondeterminism to receive a purely probabilistic execution tree. We describe how tests can be generated from a specification and applied to a black-box implementation. In addition to test functional behaviour, we assess probabilistic correctness with hypothesis testing.

An important contribution is the treatment of quiescence in a real-time environment with delays. Quiescence models the absence of outputs. If the SUT does not provide any output, a test must assess whether this behaviour is correct.

A-TEST'16, November 18, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4401-2/16/11...$15.00
http://dx.doi.org/10.1145/2994291.2994298

We summarize our main contributions:

1. the solid core of a probabilistic test theory, that incorporates stochastic time based on the **ioco** framework;
2. an operational interpretation of *quiescence* in the context of real time and exponential time-delays;
3. test case execution and evaluation as a sampling process for statistical inference on real-time behaviour.

Related Work. There is a large body of work on testing of real-time systems [2, 6, 23]. Early approaches are given by [21, 23]. Brandán-Briones et al. [6] extend the framework to incorporate the notion of quiescence. Tool support is given by Uppaal Tron [24].

Probabilistic testing preorders and equivalences are well-studied [9, 12, 34]. Prominent work by [25] introduces the concept of probabilistic bisimulation via hypothesis testing. Major influence for our work is given by [7], presenting how to observe trace probabilities during a sample process. Executable probabilistic test frameworks are defined for probabilistic finite state machines in [18, 19] and Petri nets [3].

Similar to our work is the study of Markovian bisimulation relations. Here, [13] present the foundation of an observational equivalence in form of weak bisimulation. This was refined by [11] and [35] by introducing late-weak bisimulation.

This paper is an extension of earlier work [14] investigating the test process for probabilistic automata. Novel contribution of the current work is the inclusion of stochastic time and exponential delays.

Overview over the paper. Section 2 sets the mathematical framework and introduces input-output Markov automata. In Section 3 we define the conformance relation for black-box testing with Markov automata. We study test generation and verdicts in Section 4. The paper ends with concluding remarks and plans for future work in Section 5.

2. PRELIMINARIES

We recall properties of Markov automata and show how nondeterminism is resolved. We assume that the reader is acquainted with the basics of probability theory, but recall integral definitions. See [10] for an excellent overview.

A *discrete probability distribution* over a set X is a function $\mu : X \to [0,1]$, such that $\sum_{x \in X} \mu(x) = 1$. We call μ a sub-distribution, if $\sum_{x \in X} \mu(x) \leq 1$. The set of all distributions over X is denoted $Distr(X)$ ($SubDistr(X)$ respectively).

Let Ω be a set, $\mathcal{F}_\Omega$ a σ-field of Ω and $(\Omega, \mathcal{F}_\Omega)$ the resulting measurable space. A σ-additive function $\mu : \mathcal{F}_\Omega \to [0,1]$ is called a *probability measure*, if $\mu(\Omega) = 1$.

A *probability space* is a triple $(\Omega, \mathcal{F}, Pr)$, where Ω is a set, $\mathcal{F}$ is a σ-field of Ω and $Pr : \mathcal{F} \to [0,1]$ is a probability measure, such that $Pr(\Omega) = 1$ and $Pr\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} Pr(A_i)$ for $A_i \in \mathcal{F}$, $i = 1, 2 \ldots$ pairwise disjoint.

2.1 I/O Markov Automata

Our framework is based on Markov automata [11]. Additionally, we distinguish input and output actions and follow [16] by incorporating input-reactive and output-generative transitions.

Upon receiving an input, the system decides probabilistically, which next state to move to. In contrast, a system is free to make a probabilistic choice over various output actions. Thus, each transition in an input-output Markov automaton (IOMA) either involves *one* input action and a probabilistic choice over the target states or possibly *various* output actions with their respective choice and target states.

Furthermore, a verdict must also be given to the SUT in case it produces no output at all [36]. Hence, a requirements model needs to explicitly indicate when *quiescence* is allowed. This is expressed by the special label δ.

Definition 1. *A* Markov automaton $\mathcal{M} = (S, s_0, L, \to, \rightsquigarrow)$ *is a five-tuple, consisting of*

- S *a set of states, with* s_0 *the unique starting state,*
- L *a set of actions,*
- $\to \subseteq S \times L \times Distr(S)$*, the probabilistic transition relation and*
- $\rightsquigarrow \subseteq S \times \mathbb{R}_{\geq 0} \times S$*, the Markovian transition relation.*

An IOMA is an MA, where $L = L_i \sqcup L_o \sqcup L_\tau$ *is the disjoint union of input, output and internal actions, containing a special quiescence label* $\delta \in L_o$*. It becomes input-reactive and output-generative, if we replace* $\to$ *by* $\to' \subseteq S \times Distr(L \times S)$ *with the requirement that for all* $(s, \mu) \in \to'$ *if* $\mu(s, a) > 0$ *for an input* a*, then* $\mu(s, b) = 0$ *for all* $b \neq a$*.*

An action a is enabled in state s, if there is a distribution μ, such that $(s, a, \mu) \in \to$. A state is called *probabilistic*, if at least one non-Markovian action is enabled, and *Markovian*, if at least one Markovian action is enabled. The *rate* to go from s to s' is the sum of all Markovian actions λ with $(s, \lambda, s') \in \rightsquigarrow$ and is denoted as $\mathbf{R}(s, s')$. The *exit rate* of s is the sum over all of its outgoing rates. We overload $\mathbf{R}$ by denoting $\mathbf{R}(s)$ as the exit rate of s.

The delay associated with a Markovian state is exponentially distributed with its exit rate. To illustrate, the probability to leave s_1 in Figure 1 within the next $z \in \mathbb{R}_{\geq 0}$ time units is $1 - e^{-\mathbf{R}(s_1)z} = 1 - e^{-\lambda z}$.

Note that multiple enabled Markovian actions lead to a *race condition*. The probability to move from s to its successor s' equals the probability that (one of) the Markovian timed transitions leading from s to s' wins the race. This induces the *discrete branching probability* given by

$$\mathbb{P}(s, s') := \frac{\mathbf{R}(s, s')}{\mathbf{R}(s)}.$$

A state is called *stable*, if no output or internal action is enabled. We assume *maximal progress*, meaning that time is not allowed to progress in unstable states. This renders Markovian transitions in unstable states obsolete [26].

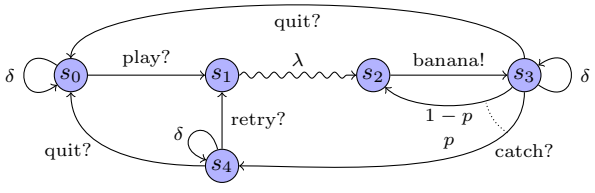


Figure 1: Toy example of an IOMA modelling a simple video game.

Example 2. Figure 1 shows an example of an IOMA. The model describes a toy model of a video game. Input is suffixed with "*?*" and output with "*!*". Discrete probability distributions are denoted with a dotted arc, together with

the action label and corresponding probabilities. Markovian actions are presented as staggered arrows.

The machine receives the input *play* to start the game. After a delay modelled by the Markovian action λ a *banana* is thrown at the user. There is a non-deterministic choice of trying to *catch* it or *quit* the game. The banana is caught according to a discrete distribution with probability $(1-p)$. If successful the process repeats, if not the user may *retry* or *quit*.

2.2 Paths and Traces

We define the language theoretic concepts for Markov automata. Let $\mathcal{M} = (S, s_0, L, \rightarrow, \rightsquigarrow)$ be an IOMA. A *path* π of $\mathcal{M}$ is a (possibly) infinite sequence of the form

$$\pi = s_1\, t_1\, \mu_1\, \alpha_1\, s_2\, t_2\, \mu_2\, \alpha_2 \ldots,$$

where $s_i \in S$, $t_i \in \mathbb{R}_{\geq 0}$, $\mu_i \in \rightarrow \cup\, \mathbb{P}_{s_i}$ and $\alpha_i \in L \cup \mathbb{R}_{\geq 0}$ for $i = 1, 2, \ldots$. We require that each finite path ends in a state. The sequence $s_i\, t_i\, \mu_i\, a_i\, s_{i+1}$ means that $\mathcal{M}$ resided t_i time units in state s_i before moving to s_{i+1} via α_i using μ_i. We denote the set of all finite paths of $\mathcal{M}$ with $\mathit{Paths}^*(\mathcal{M})$ (all paths $\mathit{Paths}(\mathcal{M})$ respectively).

An *abstract path* is a path, where each occurrence of single time values $t_i \in \mathbb{R}_{\geq 0}$ is replaced by intervals $I_i \subseteq \mathbb{R}_{\geq 0}$. Note that for any path, we can get its abstract path by replacing t_i with $[0, t_i]$. We denote all finite abstract paths of $\mathcal{M}$ with $\mathit{absPaths}^*(\mathcal{M})$ (all paths $\mathit{absPaths}(\mathcal{M})$ respectively).

The *trace* of a path $\mathit{Tr}(\pi)$ only records its visible behaviour, i.e. time and input/output actions. It is given by the (possibly) infinite sequence of the form

$$\sigma = t_1\, a_1\, t_2\, a_2 \ldots,$$

where $t_i \in \mathbb{R}_{\geq 0}$ and $a_i \in L \backslash L_\tau$ for $i = 1, 2, \ldots$. Note that a path fragment $t_1 \lambda t_2 a$ collapses to $t_1 + t_2 a$. We denote the set of all finite traces of $\mathcal{M}$ with $\mathit{traces}^*(\mathcal{M})$ (all traces $\mathit{absTraces}(\mathcal{M})$ respectively).

An *abstract trace* is given, if all $t_i \in \mathbb{R}_{\geq 0}$ are replaced by intervals $I_i \subseteq \mathbb{R}_{\geq 0}$. The set of all finite abstract traces of $\mathcal{M}$ is given by $\mathit{absTraces}^*(\mathcal{M})$ (all abstract traces $\mathit{absTraces}(\mathcal{M})$ respectively).

The operator $\mathit{act}(\pi)$ returns the *action path* of π by removing all time values t_i and distributions μ_i. For traces $\mathit{act}(\sigma)$ returns visible actions only. The *length* of a path (trace resp.) is the number of its visible actions.

2.3 Trace Distribution Semantics

Similar to the visible behaviour of labelled transition systems (LTS) being given by their traces, the visible behaviour of an IOMA is given by its trace distribution. A trace distribution is a probability space, that assigns probabilities to all abstract traces. A trace of an LTS is obtained by removing all states and internal actions from a given path. We do the same in the IOMA case: First we resolve all nondeterministic choices via an adversary and then remove all invisible information. The resolution of non-deterministic behaviour leads to a purely probabilistic structure.

The mathematical framework for infinite abstract paths is technically more involved, but completely standard [34, 39]. A classical result in measure theory shows, that it is impossible to assign a probability to all sets of traces in non-trivial scenarios. Generally, the probability assigned to a single trace is 0. To illustrate: the probability of always rolling a 6 with a die is 0, but the probability of rolling a 6 within the first 100 tries is positive. Hence, we use a cone construction of sets.

Adversaries. We follow the standard theory for probabilistic automata [34] and adapt the framework of adversaries for Markov automata. Adversaries resolve nondeterminism in an IOMA by assigning a discrete probability (sub)distribution over the set of possible outgoing distributions in every state.

We focus our considerations on adversaries that are: 1. history dependend, i.e. the adversary considers the recorded history upon scheduling the next action, 2. randomized, i.e. the adversary may make a random choice over all outgoing transitions and 3. partial, i.e. the adversary may decide to terminate the execution with a certain probability at any given point in time. Modelling termination requires us to introduce the special halting action $\perp$.

Definition 3. *A (history dependend, randomized, partial)* adversary A *of an IOMA* $\mathcal{M} = (S, s_0, L, \rightarrow, \rightsquigarrow)$ *is a function*

$$A : \mathit{Paths}^*(\mathcal{M}) \longrightarrow \mathit{SubDistr}\left(\mathit{Distr}\left(L \times S\right) \cup \perp\right),$$

such that for each finite path π *only available distributions are scheduled and output and internal actions cannot be postponed, i.e.*

- *if* $A(\pi)(\mu) > 0$, *then* $(\mathit{last}(\pi), \mu) \in \rightarrow$.
- *if* $\mathit{last}(\pi)$ *enables output or internal actions, then* $A(\pi)$ *is a full distribution.*

The value $A(\pi)(\perp)$ is the probability of *halting* the execution after π. An adversary halts after path π, if it schedules probability 1 to halting. We say an adversary halts after $k \in \mathbb{N}$ steps, if it halts for every path with length greater than k and denote this set by $\mathit{Adv}(\mathcal{M}, k)$. The set of all adversaries of an IOMA is defined as $\mathit{Adv}(\mathcal{M})$.

Intuitively, an adversary flips a multi-faced, biased coin at every step in execution to decide how to resolve nondeterminism. This results in a purely probabilistic execution. Thus, we can assign a probability to any given abstract path, via the path probability measure. The measure for a given adversary is the function $Pr_A : \mathit{absPaths}^*(\mathcal{M}) \longrightarrow [0, 1]$, inductively defined by

$$Pr_A(s_0) = 1, \text{ and } Pr_A(\Pi \cdot I\alpha s) =$$

$$Pr_A(\Pi) \cdot \begin{cases} A(\pi)(\mu) \cdot \mu(\alpha s) & \text{if } \alpha \in L \wedge I = 0 \\ \int_I \mathbf{R}(\mathit{last}(\Pi), s) \cdot e^{\mathbf{R}(s)t} dt & \text{if } \alpha \in \mathbb{R}_{\geq 0} \\ 0 & \text{otherwise,} \end{cases}$$

where π is the corresponding path to the abstract path Π.

The starting state gets assigned probability 1 and each consecutive action either gets multiplied by the probability the scheduler assigned to it or the probability that the Markovian transition takes place in the given time interval.

We quickly recall the standard cone construction for probability spaces of adversaries. For any given finite path π of length n, its cone is defined as

$$C_\pi = \left\{\pi' \in \mathit{Paths}(\mathcal{M}) \mid \pi \sqsubseteq \pi'\right\},$$

i.e. the set of all paths that have π as prefix. The set of finite paths together with the smallest σ-field generated by its set of cones and the path probability measure form a probability space on an IOMA.

Trace Distributions. A trace distribution is obtained from (the probability space) an adversary, in the way a trace is

obtained from a path; we remove all invisible behaviour. Intuitively, the probability assigned to a set of abstract traces X, is defined as the probability assigned to all abstract paths whose trace is an element of the set X.

Definition 4. *The trace distribution D of an adversary A is the probability space $(\Omega_D, \mathcal{F}_D, Pr_D)$ given by*

- $\Omega_D = traces^*(\mathcal{M})$
- *$\mathcal{F}_D$ is the smallest σ-field generated by the set of cones $\{C_\sigma \subseteq Traces(\mathcal{M}) \mid \sigma \in absTraces^*(\mathcal{M})\}$*
- *Pr_D is the unique probability measure on $\mathcal{F}_D$, such that $Pr_D(X) = Pr_A\left(Tr^{-1}(X)\right)$ for $X \in \mathcal{F}_D$*

A trace distribution is of length $k \in \mathbb{N}$, if it based on an adversary of length k. We denote the set of all such trace distributions by $trd(\mathcal{M}, k)$. The set of all trace distributions is denoted by $trd(\mathcal{M})$. This induces a natural equivalence relation $=_{TD}$. Two IOMAs are deemed equivalent, if they have the same set of trace distributions.

3. THE MAR-IOCO RELATION

The classical **ioco** reation [38] states that an implementation conforms to a specification, if it never provides any unspecified output or quiescence. For two input-output transition systems $\mathcal{I}$ and $\mathcal{S}$, with $\mathcal{I}$ *input enabled* (i.e. every input is enabled in every state), we write $\mathcal{I} \sqsubseteq \mathcal{S}$, if

$$\forall \sigma \in Traces(\mathcal{S}) : out_{\mathcal{I}}(\sigma) \subseteq out_{\mathcal{S}}(\sigma).$$

Given a requirements specification $\mathcal{S}$, the implementation $\mathcal{I}$ should not emit unforeseen behaviour. For classic transition systems, that restricts the theory to functional behaviour.

In previous work [14], we presented the probabilistic extension of **ioco** and the test process for a given probabilistic requirements specification. To generalize **ioco** to IOMA, we need two auxiliary concepts:

Trace Distribution Prefix. Given a trace distribution D of length $k+1$ and a trace distribution D of length k, we say D' is a *prefix* of D, written $D' \sqsubseteq_k D$, if all abstract traces of length k have the same probability.

Output Continuation. Given a trace distribution D of length k, its *output continuation* is the set of trace distributions assigning probability zero to abstract traces of length $k+1$ ending in inputs. This set is denoted by $outcont_{\mathcal{M}}(D)$, where $\mathcal{M}$ is the underlying IOMA.

This lets us to define the core idea of **Mar-ioco**. Intuitively, an implementation should only conform, if the probability of every trace can be matched by the specification. This includes three components: 1. functional behaviour, 2. probabilistic behaviour and 3. stochastic timing.

Definition 5. *Let $\mathcal{I}$ and $\mathcal{S}$ be IOMA with $\mathcal{I}$ input-enabled. We write $\mathcal{I} \sqsubseteq\!\!\!\!\sim \mathcal{S}$, if for all $k \in \mathbb{N}$*

$$\forall D \in trd(\mathcal{S}, k) : outcont_{\mathcal{I}}(D) \subseteq outcont_{\mathcal{S}}(D).$$

Example 6. Figure 2 displays non-conforming models according to **Mar-ioco**. An implementation is deemed non-conforming, if it emits undesired outputs, realizes wrongly implemented probabilities (e.g. an unfair coin flip, if a fair one was required) or carries out visible actions after unjustified time delays.

Note that no two of the presented examples are conforming with respect to **Mar-ioco** assuming that $\lambda \neq \nu$.

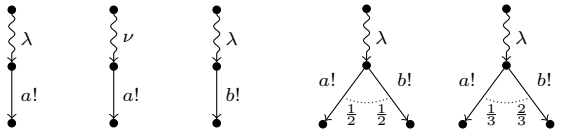


Figure 2: Examples illustrating non-conformance according to Mar-ioco.

The **Mar-ioco** relation conservatively extends the **ioco** relation to Markov automata. That is, both relations coincide for input output transition systems (IOTSs).

Theorem 7. *For two IOTSs $\mathcal{I}, \mathcal{S}$ with $\mathcal{I}$ input enabled, we have*

$$\mathcal{I} \sqsubseteq \mathcal{S} \iff \mathcal{I} \sqsubseteq\!\!\!\!\sim \mathcal{S}.$$

In **ioco** theory, the implementation is always assumed to be input enabled. If the specification is input-enabled too, **ioco** coincides with trace inclusion [38]. Assuming an input-enabled specification, our results show, that **Mar-ioco** coincides with trace distribution inclusion. Moreover, the relation is transitive, just like **ioco** [38].

Theorem 8. *Let $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ be MAs and let $\mathcal{A}$ and $\mathcal{B}$ be input enabled, then*

- *$\mathcal{A} \sqsubseteq\!\!\!\!\sim \mathcal{B}$ if and only if $\mathcal{A} \sqsubseteq_{TD} \mathcal{B}$.*
- *$\mathcal{A} \sqsubseteq\!\!\!\!\sim \mathcal{B}$ and $\mathcal{A} \sqsubseteq\!\!\!\!\sim \mathcal{B}$ imply $\mathcal{A} \sqsubseteq\!\!\!\!\sim \mathcal{C}$.*

4. TESTING FOR MA

We formalize the notion of offline tests and explain the statistical sampling process. Further, we investigate the treatment of quiescence in the presence of real-time behaviour. Lastly, we show how correct test verdicts can be assigned.

4.1 Test cases for IOMA

We consider test cases as sets of traces based on an action signature consisting of inputs and outputs (L_i, L_o). The traces describe possible behaviour of a tester. In each state of a test, the tester either provides stimuli, waits for a response of the SUT or stops the process altogether.

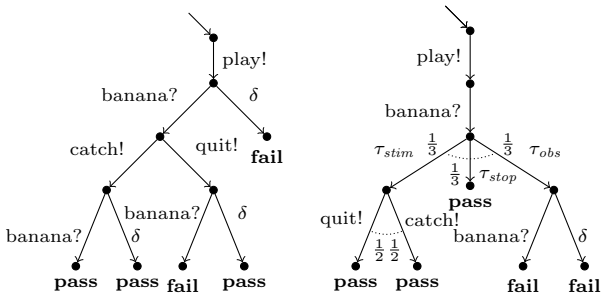
We consider tests to be IOMA without Markovian transitions, i.e. input-generative and output-reactive probabilistic automata. This enables us to model the choice between stimulating, observing and stopping probabilistically.

Definition 9. *A* test *over an action signature (L_i, L_o) is an IOMA $t = (S, s_0, L_o \setminus \{\delta\}, L_i \cup \{\delta\}, \{\tau_{stop}, \tau_{stim}, \tau_{obs}\}, \rightarrow, \emptyset)$, such that*

- *t is internally deterministic and does not contain an infinite path;*
- *t is acyclic and connected;*
- *For every state $s \in S$, either*
  - *$enabled(s) = \emptyset$*
  - *$enabled(s) = \{\tau_{stop}, \tau_{stim}, \tau_{obs}\}$*
  - *$enabled(s) = L_i \cup \{\delta\}$*
  - *$enabled(s) = L_{out}$, such that $L_{out} \subseteq L_o \setminus \{\delta\}$*

where $enabled(s)$ is the set of enabled actions in s.

The input and output sets of the test IOMA are switched to allow handshaking on shared actions. The output-generative



(a) Standard Test (b) Probabilistic Test

Figure 3: Two annotated tests derived from the specification given in Figure 1.

property of IOMAs can thus be exploited to introduce probabilistic test cases. We refer to Figure 3 for an example. To make a statement about the correctness of the observed functional behaviour, each trace of a test needs an annotation with *pass* for correct and *fail* for erroneous behaviour. The classical **ioco** test case annotation [37] suffices here.

Loosely speaking, all traces of the test, that are also present in a specification, get annotated as correct.

Example 10. Figure 3 shows two derived test cases for the requirements specification of Figure 1. Note that the action signature is mirrored to allow handshaking on shared actions.

The left side presents a standard test case for **ioco** according to [38]. After *play!* is provided to the SUT, the test waits for a response. The absence of the outputs yields the *fail* verdict, while *banana?* lets the test proceed.

The right side presents a probabilistic test case for **Mario**. After *play!* is provided and the response is *banana?* there is a uniformly distributed choice on how to proceed. If the test stimulates, there is a uniform choice over the inputs.

4.2 Test Execution and Evaluation

Since Markov automata emit probabilistic behaviour, there is a twofold evaluation process of functional and statistical correctness. While the first can be assessed via the test annotation, we illustrate the latter in the following.

Sampling. The sampling process is done by facilitating a push-button experiment in the sense of [29]. Given is a black-box timed trace machine with inputs, time and an action windows and a reset button as illustrated in Figure 4.

We choose a sample length $k \in \mathbb{N}$ and width $m \in \mathbb{N}$, i.e. how long shall we observe a single run and how many runs should be observed. Additionally, we choose a level of significance $\alpha \in (0,1)$, e.g. $\alpha = 0.05$ or $\alpha = 0.01$ are common in practice. We assume that the timer resets to 0 after every visible action and that two consecutive occurrences of the same action are distinguishable. An external observer records each individual execution before the reset button is pressed and the machine starts again. Thus, we collect m traces of length k, which are summarized as a *sample* O.

During each run the black-box is governed by a trace distribution $D \in trd(\mathcal{I})$. For simplicity, we assume that D is the same in every execution. The SUT makes two choices: 1. choosing a trace distribution D and 2. D chooses a trace.

Frequencies and expectations. We measure the deviation of the sample distribution to the expected distribution, given

Figure 4: A timed trace machine with reset button, inputs $a_0?, \ldots, a_z?$ and time and action windows.

by the specification. The latter is given for any underlying trace distribution D. Hence, the expected probability to observe an abstract trace σ is given by $Pr_D(\sigma)$.

Depending on the accuracy of time measurement, it is unlikely to record the same *timed* trace more than once, c.f. Figure 5. Therefore, we group traces in classes based on the same visible action behaviour. For a given abstract trace σ, its class Σ_σ is the set of all abstract traces $\varrho \in O$, such that $act(\sigma) = act(\varrho)$. A sample of length k and width m then induces a frequency measure, given by

$$freq_O(\sigma) = \frac{|\Sigma_\sigma|}{m} \Pi_{i=1}^{k} \frac{|\{\varrho \in \Sigma_\sigma \mid I_i^\varrho \subseteq I_i^\sigma\}|}{|\Sigma_\sigma|}$$

Probabilistic choices of the SUT make inference on time-delays challenging. To illustrate, assume a sample is given by the two traces $\sigma_1 = 0\,a\,1\,b$, and $\sigma_2 = 1\,a\,0\,b$. The underlying parameters of the SUT are unknown and the only available information is the sample. To infer about the probability to observe $\varrho = 0.5\,a\,0.5, b$, we resort to non-parametric statistical inference and apply order statistics on the observed times [15]. This results in the above frequency measure.

Treatment of quiescence. A test case needs to assess if an SUT is allowed to be unresponsive when output was expected [36]. Since quiescence represents the absence of output for indefinite time, it should be regarded with attention in practical test scenarios. Earlier work assumes a global fixed time-out value set by a user [6].

Since progress of Markov automata may be exponentially delayed, a global time-out value has two disadvantages: 1. a time-out might occur, before a specified Markovian action takes place and 2. a global time-out value might unnecessarily prolong the test process. Therefore, our interest is to minimize the probability of erroneously declaring quiescence, while keeping the overall testing time as low as possible.

Assume a level of significance $\alpha \in (0,1)$ is given. Let λ be the exit rate of a state s. Then the exit rate of s is a random variable T that is exponentially distributed with parameter λ. The probability, that a Markovian action is executed before a state-specific maximum waiting time t^s_{max} expires should be greater than $(1-\alpha)$, i.e.

$$P(T < t^s_{max}) > 1 - \alpha$$

Hence, choosing $t^s_{max} > -\frac{\log \alpha}{\lambda}$ minimizes the probability of assigning quiescence, when the SUT actually makes progress.

This approach does not cover consecutive Markovian transitions or divergent behaviour. This is left for future work.

Example 11. Figure 5 shows a possible sample obtained from the video game modelled in Figure 1. It contains 10 traces of length 4. Neglecting time information yields three different trace classes Σ_{σ_1}, Σ_{σ_5} and Σ_{σ_9} based on $act(\sigma_1) = \ldots = act(\sigma_4)$, $act(\sigma_5) \ldots act(\sigma_8)$ and $act(\sigma_9)$.

id	Recorded Trace σ	n_σ
σ_1	0 play 2.2 banana 0 catch 0 banana	1
σ_2	0 play 2.4 banana 0 catch 0 banana	1
σ_3	0 play 2.5 banana 0 catch 0 banana	1
σ_4	0 play 2.6 banana 0 catch 0 banana	1
σ_5	0 play 2.3 banana 0 quit 5 δ	1
σ_6	0 play 2.5 banana 0 quit 5 δ	1
σ_7	0 play 2.6 banana 0 quit 5 δ	1
σ_8	0 play 4.0 banana 0 quit 5 δ	1
σ_9	5 δ 5 δ 5 δ 5 δ	2

Figure 5: Possible sample of the model in Figure 1 containing 10 traces of length 4 and the number of their occurrences.

The frequency, with which to expect a delay of 2.5 time units before seeing *banana* followed by *catch* and *banana* based on the sample is given as $\frac{3}{10}$.

Moreover, note that the time-out value in s_0 is set to 5 time units, resulting in the observation of two identical quiescent traces in σ_9.

Acceptable Outcomes. A sample O is accepted, if $freq_O$ lies within some distance r of the expected distribution Pr_D. Recall the definition of a closed ball centred at $x \in X$ with radius r is given by $B_r(r) = \{y \in X \mid dist(x,y) \leq r\}$. All measures that deviate at most r from the expected measure Pr_D are contained in the ball $B_r(r)$, where

$$dist(\mu,\nu) = \sup_{\sigma \in absTraces^*(\mathcal{M})} |\mu(\sigma) - \nu(\sigma)|$$

is the total variation distance of two measures. In order to minimize the error of falsely accepting a sample, we choose the smallest radius $\bar{r}$, such that the error of falsely rejecting a sample is not greater than the level of significance $\alpha \in (0,1)$. Every sample, that lies within distance $\bar{r}$ to the expected measure is hence deemed as an *acceptable outcome* under D. We denote this set by $Obs(D,\alpha,k,m)$. The entire set of *observations* of $\mathcal{M}$ is given by

$$Obs(\mathcal{M},\alpha,k,m) = \bigcup_{D \in trd(\mathcal{M},k)} Obs(D,\alpha,k,m).$$

Our goal is to find $D \in trd(\mathcal{S})$ that maximises the likelihood of a given *sample* O of the SUT. If the frequencies obtained from O have a probability of more than $(1-\alpha)$ under the measure induced by the maximum-likelihood distribution D, we have no statistical evidence to reject the SUT. We denote this probability as $\mathbb{P}_{\mathcal{S}}(O)$.

Verdicts. The following process decides if an implementation fails for functional and/or statistical behaviour.

Given are a requirements specification $\mathcal{S}$, an annotated test $\widehat{t}$ for $\mathcal{S}$, parameters $\alpha \in (0,1)$ and $k,m \in \mathbb{N}$, where k is given by the test length. We define the *functional verdict* as

$$v_{\widehat{t}}(\mathcal{I}) = \begin{cases} pass & \text{if all tested traces have } pass \text{ annotation} \\ fail & \text{otherwise,} \end{cases}$$

and the *statistical verdict* as

$$v_t^{\alpha}(\mathcal{I}) = \begin{cases} pass & \text{if } \mathbb{P}_{\mathcal{S}}(Obs(\mathcal{I} \,||\, t,\alpha,k,m)) \geq 1-\alpha \\ fail & \text{otherwise,} \end{cases}$$

where $\mathcal{I} \,||\, t$ denotes the parallel composition of test and SUT.

An implementation passes the functional verdict, if the annotation of every observed trace is labelled *pass*. It passes the statistical verdict, if there is an adversary of the specification, that makes the sampled behaviour likely. Lastly, it passes the test suite, if it passes both functional and statistical verdict for each test case in the test suite.

4.3 Soundness and Completeness

The correctness of the framework is formalized as soundness and completeness (a.k.a. exhaustiveness). *Soundness* ensures that test cases assign the correct verdict. *Completeness* postulates that the framework is powerful enough to discover each deviation from the specification.

Since the underlying model is probabilistic, there remains a degree of uncertainty known as the *errors of first and second kind*. For MBT of probabilistic systems this translates to the likelihood to reject a correct implementation and to accept an erroneous one. Hence, a test suite can only be considered sound and complete with a guaranteed (high) probability.

At the time of writing this paper, soundness and completeness of the framework are left as conjectures. There is evidence suggesting that these properties do hold [14].

Soundness expresses for a given $\alpha \in (0,1)$, that there is a $(1-\alpha)$ probability, that a correct system passes the test suite for sufficiently large sample width m.

Conjecture 12. *Each annotated test for an IOMA $\mathcal{S}$ is sound for every level of significance $\alpha \in (0,1)$ with respect to* ***Mar-ioco****.*

Completeness of a test suite is inherently a theoretic result. Possible loops and infinite behaviour in an SUT, require a test suite of infinite size. Additionally, there is the chance of falsely accepting an erroneous implementation due to the statistical character of our framework. However, the latter is bound from above and decreases with bigger sample size.

Conjecture 13. *The set of all annotated tests for an IOMA $\mathcal{S}$ is complete for every level of significance $\alpha \in (0,1)$ with respect to* ***Mar-ioco****.*

5. CONCLUSIONS

We described our ongoing efforts in developing a model-based test framework that incorporates continuous time and probabilistic behaviour, based on an **ioco**-style structure. We defined the conformance relation **Mar-ioco** stating what it means for an SUT to pass a test suite. We described the advantage of state-specific waiting times to reduce the time needed for the overall test process. Additionally, we described the statistical sampling process and how correct verdicts can be assigned with a high probability assuming a sufficient sample size.

There is ample future work: A first step is proving the correctness of the framework. Quiescence and divergence in the presence of consecutive exponentially distributed delays should be further investigated. Moreover, we plan to find more sophisticated inference methods to assign a statistical verdict. Lastly, we aim at applying the theory in a real life case study.

6. REFERENCES

[1] J. N. Al-Karaki and A. E. Kamal. Routing Techniques in Wireless Sensor Networks: A Survey. *IEEE Wireless Commun.*, 11(6):6–28, 2004.

[2] H. Bohnenkamp and A. Belinfante. Timed Testing with TorX. In *Formal Methods Europe*, volume 3582 of *LNCS*, pages 173–188. Springer, 2005.

[3] F. Böhr. Model-Based Statistical Testing of Embedded Systems. In *IEEE 4th Intl. Conf. on Software Testing, Verification and Validation*, pages 18–25, 2011.

[4] M. Bozga, A. David, A. Hartmanns, H. Hermanns, K. G. Larsen, A. Legay, and J. Tretmans. State-of-the-Art Tools and Techniques for Quantitative Modeling and Analysis of Embedded Systems. In *DATE*, pages 370–375, 2012.

[5] M. Bozzano, A. Cimatti, J. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. Safety, Dependability and Performance Analysis of Extended AADL Models. *Comput. J.*, 54(5):754–775, 2011.

[6] L. B. Briones and E. Brinksma. A Test Generation Framework for Quiescent Real-Time Systems. In *4th International Workshop FATES*, pages 64–78, 2004.

[7] L. Cheung, M. Stoelinga, and F. Vaandrager. A Testing Scenario for Probabilistic Processes. *ACM*, 54(6), 2007.

[8] S. Choi, D. Dachman-Soled, T. Malkin, and H. Wee. Improved Non-Committing Encryption with Applications to Adaptively Secure Protocols. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 287–302. Springer, 2009.

[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] D. L. Cohn. *Measure Theory*. Birkhäuser, 1980.

[11] Y. Deng and M. Hennessy. On the Semantics of Markov Automata. *Inf. and Computation*, 222:139–168, 2013.

[12] Y. Deng, M. Hennessy, R. J. van Glabbeek, and C. Morgan. Characterising Testing Preorders for Finite Probabilistic Processes. *CoRR*, 2008.

[13] C. Eisentraut, H. Hermanns, and L. Zhang. On Probabilistic Automata in Continuous Time. In *IEEE 25th Annual Symposium on LICS*, pages 342–351, 2010.

[14] M. Gerhold and M. Stoelinga. *Model-Based Testing of Probabilistic Systems*, pages 251–268. FASE. Springer, 2016.

[15] J. Gibbons and S. Chakraborti. *Nonparametric Statistical Inference 4th Ed.* Taylor & Francis, 2014.

[16] R. J. v. Glabbeek, S. A. Smolka, B. Steffen, and C. Tofts. Reactive, Generative, and Stratified Models of Probabilistic Processes. pages 130–141. IEEE Computer Society Press, 1990.

[17] H. Hermanns. *Interactive Markov Chains: And the Quest for Quantified Quality*. Springer, 2002.

[18] R. M. Hierons and M. G. Merayo. Mutation Testing from Probabilistic and Stochastic Finite State Machines. *Journal of Systems and Software*, pages 1804–1818, 2009.

[19] I. Hwang and A. R. Cavalli. Testing a Probabilistic FSM using Interval Estimation. *Computer Networks*, pages 1108–1125, 2010.

[20] C. Jegourel, A. Legay, and S. Sedwards. *A Platform for High Performance Statistical Model Checking - PLASMA*. Springer, 2012.

[21] M. Krichen and S. Tripakis. Conformance Testing for Real-Time Systems. *Formal Methods in System Design*, 34(3):238–304, 2009.

[22] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 200–204. Springer, 2002.

[23] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online Testing of Real-Time Systems using UPPAAL. volume 3395 of *LNCS*, pages 79–94. Springer, 2005.

[24] K. G. Larsen, M. Mikucionis, and B. Nielsen. Uppaal Tron User Manual. *BRICS, Aalborg Univ.*, 2009.

[25] K. G. Larsen and A. Skou. Bisimulation Through Probabilistic Testing. pages 344–352. ACM Press, 1989.

[26] M. Lohrey, P. R. D'Argenio, and H. Hermanns. Axiomatising Divergence. In *International Colloquium on Automata, Languages, and Programming*, pages 585–596. Springer, 2002.

[27] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., 1994.

[28] M. A. Marsan, G. Conte, and G. Balbo. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, 1984.

[29] R. Milner. *A Calculus of Communicating Systems*. 1980.

[30] R. D. Nicola and M. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34(1):83 – 133, 1984.

[31] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[32] A. Remke and M. Stoelinga, editors. *Stochastic Model Checking*, volume 8453 of *LNCS*. Springer, 2014.

[33] E. Ruijters and M. Stoelinga. Fault Tree Analysis: A Survey of the State-of-the-Art in Modeling, Analysis and Tools. *Computer Science Review*, 15:29–62, 2015.

[34] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Cambridge, MA, USA, 1995.

[35] L. Song, L. Zhang, J. C. Godskesen, H. Hermanns, and C. Eisentraut. Late Weak Bisimulation for Markov Automata. *CoRR*, abs/1202.4116, 2012.

[36] W. G. J. Stokkink, M. Timmer, and M. Stoelinga. Divergent Quiescent Transistion Sytems. In *7th Conf. on Tests and Proofs (TAP)*, LNCS, 2013.

[37] M. Timmer, H. Brinksma, and M. Stoelinga. Model-Based Testing. In *Software and Systems Safety: Specification and Verification*, volume 30 of *NATO Science for Peace and Security*, pages 1–32. IOS Press, 2011.

[38] J. Tretmans. Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.

[39] V. Wolf, C. Baier, and M. Majster-Cederbaum. Trace Semantics for Stochastic Systems with Nondeterminism. *ENTCS*, 164(3):187 – 204, 2006. Proc. of the 4th Int. Workshop on Quantitative Aspects of Programming Languages, QAPL.