Model-Based Testing of Probabilistic Systems

Marcus Gerhold and Mariëlle Stoelinga(✉)

University of Twente, Enschede, The Netherlands
m.gerhold@utwente.nl, marielle@cs.utwente.nl

Abstract. This paper presents a model-based testing framework for probabilistic systems. We provide algorithms to generate, execute and evaluate test cases from a probabilistic requirements model. In doing so, we connect *ioco*-theory for model-based testing and statistical hypothesis testing: our *ioco*-style algorithms handle the functional aspects, while statistical methods, using χ^2 tests and fitting functions, assess if the frequencies observed during test execution correspond to the probabilities specified in the requirements.

Key results of our paper are the classical soundness and completeness properties, establishing the mathematical correctness of our framework; Soundness states that each test case is assigned the right verdict. Completeness states that the framework is powerful enough to discover each probabilistic deviation from the specification, with arbitrary precision.

We illustrate the use of our framework via two case studies.

1 Introduction

Probability. Probability plays an important role in many computer applications. A vast number of randomized algorithms, protocols and computation methods use randomization to achieve their goals. Routing in sensor networks, for instance, can be done via random walks [1]; speech recognition is based on hidden Markov models [32]; population genetics use Bayesian computation [2], security protocols use random bits in their encryption methods [10], control policies in robotics, leading to the emerging field of probabilistic robotics, concerned with perception and control in the face of uncertainty networking algorithms assign bandwidth in a random fashion. Such applications can be implemented in one of the many probabilistic programming languages, such as Probabilistic-C [26] or Figaro [28]. At a higher level, service level agreements are formulated in a stochastic fashion, stating that the average uptime should be at least 99 %, or that the punctuality of train services should be 95 %.

Key question is whether such probabilistic systems are correct: is bandwidth distributed fairly among all parties? Is the up-time, packet delay and jitter according to specification? Do the trains on a certain day run punctual

This work has been supported by the NWO project BEAT (612.001.303), the STW project SUMBAT (13859), the EU FP7 project SENSATION (318490) and by the STW and ProRail project ArRangeer (12238).

© Springer-Verlag Berlin Heidelberg 2016
P. Stevens and A. Wąsowski (Eds.): FASE 2016, LNCS 9633, pp. 251–268, 2016.
DOI: 10.1007/978-3-662-49665-7_15

enough? To investigate such questions, probabilistic verification has become a mature research field, putting forward models like probabilistic automata (PAs) [33,38], Markov decision processes [30], (generalized) stochastic Petri nets [23], with verification techniques like stochastic model checking [31], and tools like Prism [20].

Testing. In practice, the most common validation technique is testing, where we subject the system under test to many well-designed test cases, and compare the outcome to the specification. Surprisingly, only few papers are concerned with the testing of probabilistic systems[1], with notable exceptions being [16,18].

This paper presents a model-based testing framework for probabilistic systems. Model-based testing (MBT) is an innovative method to automatically generate, execute and evaluate test cases from a requirements model. By providing faster and more thorough testing at lower cost, MBT has gained rapid popularity in industry. A wide variety of MBT frameworks exist, capable of handling different system aspects, such as functional properties [40], real-time [5,8,22], quantitative aspects [7], and continuous behaviour [27]. As stated, MBT approaches dealing with probability are underdeveloped.

Our Approach. Our specification is given as *probabilistic input/output transition system pIOTS,* a mild generalization of the PA model. As usual, pIOTSs contain two type of choices, *non-deterministic choices* model choices that are not under the control of the system. As argued in [33], these are needed to model phenomena like implementation freedom, scheduler choices, intervals of probability and interleaving. *Probabilistic choices* model random choices made by the system (e.g., coin tosses) or nature (e.g., failure probabilities, degradation rates).

Important contribution are our algorithms to automatically generate, execute and evaluate test cases from a specification pIOTS. These test cases are probabilistic and check if both the functional and the probabilistic behaviour conform to the specification. Probability is observed through frequencies, hence we execute each test multiple times. We use statistical hypothesis testing, in particular the χ^2 test, to assess whether a test case should pass or fail. Technical complication here is the non-determinism in pIOTSs, which prevents us from directly using the χ^2 test. Rather, we first need to find the best resolution of the non-determinism that could have led to these observations. To do so, we set up a non-linear optimization problem that finds the best fit for the χ^2 test.

Key result of our paper is the soundness and completeness of our framework. *Soundness* states that each test case we derive contains the correct verdict: a pass if the behaviour observed during testing conforms to the requirements; a fail if it does not. *Completeness* states that the framework is powerful enough to discover each deviation of non-conforming implementations. Formulating the soundness and completeness results requires a formal notion of conformance. Here, we propose the *pioco*-relation, which pins down when an implementation

[1] Note that the popular topic of statistical testing is concerned with choosing the test inputs probabilistically; it does not check for the correctness of the random choices made by a system itself.

modelled as pIOTS conforms to a specification pIOTs. We prove several properties of the *pioco*-relation, in particular it being a conservative extension of *ioco*. Lastly, we illustrate our approach with two case studies: the exponential binary back off protocol, and the IEEE 1394 root contention protocol.

While test efficiency is important, this paper focusses on the methodological set up and correctness. Important future work is to optimize the statistical verdicts we derive and to provide a fully fledged implementation of our methods.

Related Work. Probabilistic testing preorders and equivalences are well studied [11,13,34], defining when two probabilistic transition systems are equivalent, or one subsumes the other. In particular, early and influential work by [21] introduces the fundamental concept of probabilistic bisimulation via hypothesis testing. Also, [9] shows how to observe trace probabilities via hypothesis testing. Executable test frameworks for probabilistic systems have been defined for probabilistic finite state machines [17,24], dealing with mutations and stochastic timing, Petri nets [6], and CSL [35,36]. The important research line of statistical testing [4,42,43] is concerned with choosing the inputs for the SUT in a probabilistic way in order to optimize a certain test metric, such as (weighted) coverage. The question on when to stop statistical testing is tackled in [29].

An approach similar in the spirit of ours is by Hierons et al. [16]. However, our model can be considered as an extension of [16] reconciling probabilistic and non-deterministic choices in a fully fledged way. Being more restrictive enables [16] to focus on individual traces, whereas we use trace distributions.

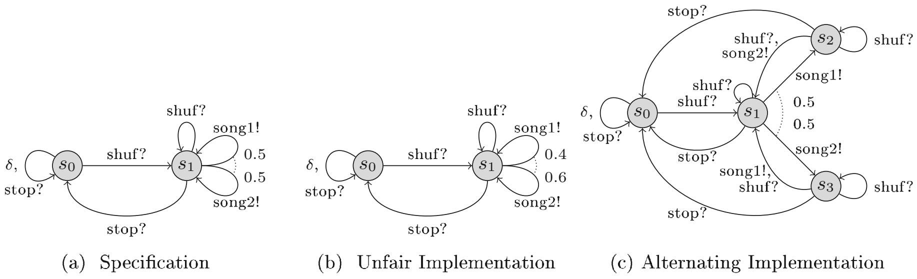
Furthermore, the current paper extends a workshop paper by [14] that introduced the *pioco*-relation and roughly sketched the test case process. Novel contributions of our current paper are 1. a more generic model pIOTS model that includes internal transitions, 2. the soundness and completeness results, 3. solid definitions of test cases, test execution, and verdicts, 4. the treatment of quiescence, i.e., absence of outputs, 5. the handling of probabilistic test cases.

Overview of the Paper. Section 2 sets the mathematical framework and introduces pIOTSs, adversaries and trace distributions. Section 3 shows how we generate and execute probabilistic tests and evaluate them functionally and statistically. Section 4 introduces the *pioco* relation and shows the soundness and completeness of our testing method. Two case studies can be found in Sect. 5. Lastly Sect. 6 ends the paper with future work and conclusions.

2 Preliminaries

2.1 Probabilistic Input/Output Systems

We start by introducing some standard notions from probability theory. A *discrete probability distribution* over a set X is a function $\mu : X \longrightarrow [0,1]$ such that $\sum_{x \in X} \mu(x) = 1$. The set of all distributions over X is denoted by $Distr(X)$. The probability distribution that assigns 1 to a certain element $x \in X$ is called the *Dirac* distribution over x and is denoted $Dirac(x)$.



(a) Specification (b) Unfair Implementation (c) Alternating Implementation

Fig. 1. Specification and two implementations of a shuffle music player. Actions separated by commas indicate that two transitions are enabled from the state.

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

Definition 1. *A* probabilistic input/output transition system *is a sixtuple* $\mathcal{A} = (S, s, L_I, L_O, L_H, \Delta)$*, where*

- S *is a finite set of states,*
- s_0 *is the unique starting state,*
- L_I*,* L_O *and* L_H *are disjoint sets of input, output and internal labels respectively, containing a special quiescence label* $\delta \in L_O$*. We write* $L = L_I \cup L_O^{\delta} \cup L_H$ *for the set of all labels.*
- $\Delta \subseteq S \times Distr(L \times S)$ *a finite transition relation such that for all input actions* $a?$*,* $\mu(a?, s') > 0$ *implies* $\mu(b, s'') = 0$ *for all* $b \neq a?$*.*

We use "?" to suffix input and "!" to suffx output. We write $s \xrightarrow{\mu,a} s'$ *if* $(s, \mu) \in \Delta$ *and* $\mu(a, s') > 0$*; and* $s \to a$ *if there are* $\mu \in Distr(L \times S)$ *and* $s' \in S$ *such that* $s \xrightarrow{\mu,a} s'$ *(*$s \not\to a$ *if not). We write* $s \xrightarrow{\mu,a}_{\mathcal{A}} s'$*, etc. to clarify ambiguities if needed. Lastly,* $\mathcal{A}$ *is* input-enabled *if for all* $s \in S$ *we have* $s \to a?$ *for all* $a \in L_I$*.*

Following [15], pIOTSs are *input-reactive* and *output-generative*. Upon receiving an input, the pIOTS decides probabilistically which next state to move to. On producing an output, the pIOTS chooses both the output action and the state probabilistically. As required in clause 4 of Definition 1, this means that each transition can either involve a single input action, or several outputs, quiescence or internal actions. Note that a state can enable input and output transitions albeit not in the same distribution. Furthermore, in testing, a verdict must also be given if the system-under-test is quiescent, i.e., produces no output at all. Hence, the requirements model must explicitly indicate when quiescence is allowed, which is expressed by a special output label δ, for details see [39,41].

Example 2. Figure 1 shows three models of a simple shuffle mp3 player with two songs. The pIOTS in (Fig. 1a) models the requirements: pressing the shuffle button enables the two songs with probability 0.5 each, repeatedly until stop is pressed.

Implementation (Fig. 1b) is subject to a small probabilistic deviation. In implementation (Fig. 1c) the same song cannot be played twice in a row without intervention of the shuffle button. States without enabled output transition allow quiescence, denoted by δ transitions. The model-based testing framework established in the paper is capable of detecting all of the above flaws.

Parallel composition is defined in a standard fashion. Two pIOTSs in composition synchronize on shared actions, and evolve independently on others. Since the transitions in the component pIOTSs are stochastically independent, we multiply the probabilities when taking shared actions, denoted by $\mu \times \nu$. To avoid name clashes, we only compose compatible pIOTSs. Note that parallel composition of two input-enabled pIOTSs yields a pIOTS.

Definition 3. *Two pIOTSs* $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$ *and* $\mathcal{A}' = (S', s_0', L_I', L_O', L_H', \Delta')$, *are* compatible *if* $L_O \cap L_O' = \{\delta\}$, $L_H \cap L' = \emptyset$ *and* $L \cap L_H' = \emptyset$. *Their* parallel composition *is the tuple*

$$\mathcal{A} \,||\, \mathcal{A}' = \left(S'', (s_0, s_0'), L_I'', L_O'', L_H'', \Delta''\right), \textit{ where}$$

$S'' = S \times S'$, $L_I'' = (L_I \cup L_I') \setminus (L_O \cup L_O')$, $L_O'' = L_O \cup L_O'$, $L_H'' = L_H \cup L_H'$, *and finally* $\Delta'' = \{((s,t), \mu) \in S'' \times \mathit{Distr}\,(L'' \times S'') \mid$

$$\mu \equiv \begin{cases} \nu_1 \times \nu_2 & if\, \exists a \in L \cap L' \, such\, that\, s \xrightarrow{\nu_1, a} \wedge t \xrightarrow{\nu_2, a} \\ \nu_1 \times \mathbb{1} & if\, \forall a \in L\, with\, s \xrightarrow{\nu_1, a} \, we\, have\, t \not\xrightarrow{} a \\ \mathbb{1} \times \nu_2 & if\, \forall a \in L'\, with\, t \xrightarrow{\nu_2, a} \, we\, have\, s \not\xrightarrow{} a \end{cases} \},$$

where $(s, \nu_1) \in \Delta$,$(t, \nu_2) \in \Delta'$ *respectively, and* $\nu_1 \times \mathbb{1}\left((s,t), a\right) = \nu_1(s,a) \cdot 1$ *and* $\mathbb{1} \times \nu_2\left((s,t), a\right) = 1 \cdot \nu_2(t,a)$.

2.2 Paths and Traces

We define the usual language concepts for LTSs. Let $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$ be a pIOTS. A *path* π of $\mathcal{A}$ is a (possibly) infinite sequence of the following form

$$\pi = s_1 \mu_1 a_1 s_2 \mu_2 a_2 s_3 \mu_3 a_3 s_4 \ldots,$$

where $s_i \in S$, $a_i \in L$ and $\mu_i \in \mathit{Distr}\,(L \times S)$, such that each finite path ends in a state and $s_i \xrightarrow{\mu_{i+1}, a_{i+1}} s_{i+1}$ for each non-final i. We use $\mathit{last}\,(\pi)$ to denote the last state of a finite path ($\mathit{last}\,(\pi) = \infty$ for infinite paths). The set of all finite paths of $\mathcal{A}$ is denoted by $\mathit{Path}^*(\mathcal{A})$ and all infinite paths by $\mathit{Path}\,(\mathcal{A})$.

The associated *trace* of a path π is obtained by omitting states, distributions and internal actions, i.e. $\mathit{trace}\,(\pi) = a_1 a_2 a_3 \ldots$. Conversely, $\mathit{trace}^{-1}(\sigma)$ gives the set of all paths, which have trace σ. The *length* of a path is the number of occurring actions on its associated trace. All finite traces of $\mathcal{A}$ are summarized in $\mathit{traces}\,(\mathcal{A})$. The set of *complete traces*, $\mathit{ctraces}\,(\mathcal{A})$, contains every trace based on paths ending in states that do not enable any more actions. We write $\mathit{out}_{\mathcal{A}}(\sigma)$ for the set of all output actions enabled with positive probability after trace σ.

2.3 Adversaries and Trace Distributions

Very much like traces of LTSs are obtained by first selecting a path and by then removing all states and internal actions, we do the same in the probabilistic case. First, we resolve all non-deterministic choices in the pIOTS via an adversary and then we remove all states to get the trace distribution.

The resolution of the non-determinism via an adversary leads to a purely probabilistic system, in which we can assign a probability to each finite path. A classical result in measure theory [12] shows that it is impossible to assign a probability to all sets of traces, hence we use σ-fields $\mathcal{F}$ consisting of cones.

Adversaries. Following the standard theory for probabilistic automata [38], we define the behaviour of a pIOTS via adversaries (a.k.a. policies or schedulers) to resolve the non-deterministic choices in pIOTSs; in each state of the pIOTS, the adversary may choose which transition to take or it may also halt the execution.

Given any finite history leading to a state, an adversary returns a discrete probability distribution over the set of next transitions. In order to model termination, we define schedulers such that they can continue paths with a halting extension, after which only quiescence is observed.

Definition 4. *An* adversary E *of a pIOTS* $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$ *is a function*

$$E : Path^* (\mathcal{A}) \longrightarrow Distr\left(Distr\left(L \times S\right) \cup \{\perp\}\right),$$

such that for each finite path π*, if* $E(\pi)(\mu) > 0$*, then* $(last(\pi), \mu) \in \Delta$ *or* $\mu \equiv \perp$*. We say that* E *is deterministic, if* $E(\pi)$ *assigns the Dirac distribution to every distribution after all* $\pi \in Path^*(\mathcal{A})$*. The value* $E(\pi)(\perp)$ *is considered as interruption/halting. An adversary* E *halts on a path* π*, if* $E(\pi)(\perp) = 1$*. We say that an adversary halts after* $k \in \mathbb{N}$ *steps, if it halts for every path of length greater or equal to* k*. We denote all such finite adversaries by* $adv(\mathcal{A}, k)$*.*

Intuitively an adversary tosses a multi-faced and biased die at every step of the computation, thus resulting in a purely probabilistic computation tree. The probability assigned to a path π is obtained by the probability of its cone $C_\pi = \{\pi' \in Path(\mathcal{A}) \mid \pi \sqsubseteq \pi'\}$. We use the inductively defined path probability function Q^E, i.e. $Q^E(s_0) = 1$ and $Q^E(\pi\mu a s) = Q^E(\pi) E(\pi)(\mu) \mu(a, s)$. This function enables us to assign a unique probability space $(\Omega_E, \mathcal{F}_E, P_E)$ associated to an adversary E. Thus, the probability of π is $P_E(\pi) = P_E(C_\pi) = Q^E(\pi)$.

Trace Distributions. A trace distribution is obtained from (the probability space of) an adversary by removing all states. Thus, the probability assigned to a set of traces X is the probability of all paths whose trace is an element of X.

Definition 5. *The* trace distribution H *of an adversary* E*, denoted* $H = trd(E)$ *is the probability space* $(\Omega_H, \mathcal{F}_H, P_H)$ *given by*

1. $\Omega_H = L^\omega$,
2. $\mathcal{F}_H$ *is the smallest* σ*-field containing the set* $\{C_\beta \subseteq \Omega_H \mid \beta \in L^\omega\}$,
3. P_H *is the unique prob. measure on* $\mathcal{F}_H$ *such that* $P_H(X) = P_E\left(trace^{-1}(X)\right)$ *for* $X \in \mathcal{F}_H$*.*

We write $trd(\mathcal{A})$ for the set of all trace distributions of $\mathcal{A}$ and $trd(\mathcal{A}, k)$ for those halting after $k \in \mathbb{N}$. Lastly we write $\mathcal{A} =_{TD} \mathcal{B}$ if $trd(\mathcal{A}) = trd(\mathcal{B})$, $\mathcal{A} \sqsubseteq_{TD} \mathcal{B}$ if $trd(\mathcal{A}) \subseteq trd(\mathcal{B})$ and $\mathcal{A} \sqsubseteq_{TD}^{k} \mathcal{B}$ if $trd(\mathcal{A}, k) \subseteq trd(\mathcal{B}, k)$ for $k \in \mathbb{N}$.

The fact that $(\Omega_E, \mathcal{F}_E, P_E)$ and $(\Omega_H, \mathcal{F}_H, P_H)$ define probability spaces, follows from standard measure theory arguments (see for example [12]).

Example 6. Consider (c) in Fig. 1 and an adversary E starting from the beginning state s_0 scheduling probability 1 to shuf?, 1 to the distribution consisting of song1! and song2! and $\frac{1}{2}$ to both shuffle? transitions in s_2. Then choose the paths $\pi = s_0\mu_1\text{shuf?}s_1\mu_2\text{song1!}s_2\mu_3\text{shuf?}s_2$ and $\pi' = s_0\mu_1\text{shuf?}s_1\mu_2\text{song1!}s_2\mu_4\text{shuf?}s_1$.

We see that $\sigma = trace(\pi) = trace(\pi')$ and $P_E(\pi) = Q^E(\pi) = \frac{1}{4}$ and $P_E(\pi') = Q^E(\pi') = \frac{1}{4}$, but $P_{trd(E)}(\sigma) = P_E\left(trace^{-1}(\sigma)\right) = P_E(\{\pi, \pi'\}) = \frac{1}{2}$.

3 Testing with pIOTS

3.1 Test Generation

Model-based testing entails the automatic test case generation, execution and evaluation based on a requirements model. We provide two algorithms for test case generation: an *offline* or *batch* algorithm that generates test cases before their execution; and an *online* or *on-the-fly* algorithm generating test cases during execution.

First, we formalize the notion of a (offline) test case over an action signature (L_I, L_O). In each state of a test, the tester can either provide some stimulus $a? \in L_I$, or wait for a response of the system or stop the testing process.[2] Each of these possibilities can be chosen with a certain probability, leading to probabilistic test cases. We model this as a probabilistic choice between the internal actions τ_{obs}, τ_{stop} and τ_{stim}. Note that, even in the non-probabilistic case, the test cases are often generated probabilistically in practice, but this is not supported in theory. Thus, our definition fills a small gap here.

Furthermore, note that, when waiting for a system response, we have to take into account all potential outputs in L_O, including the situation that the system provides no response at all, modelled by δ. Since the continuation of a test depends on the history, offline test cases are formalized as trees.

Definition 7. *A* test *or* test case *over an action signature* (L_I, L_O) *is a pIOTS of the form* $t = (S, s_0, L_O \setminus \{\delta\}, L_I \cup \{\delta\}, \{\tau_{obs}, \tau_{stim}, \tau_{stop}\}, \Delta)$ *such that*

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

[2] Note that in more recent version of *ioco* theory [41], test cases are input-enabled. This can easily be incorporated into our framework.

where $after\,(s)$ *is the set of actions in state* s*. A* test suite T *is a set of test cases. A test case (suite) for a pIOTS* $\mathcal{A}_{\mathcal{S}} = (S, s_0, L_I, L_O, L_H, \Delta)$*, is a test case (suite) over* (L_I, L_O)*.*

Note that the action signature of tests has switched input and output label sets.

Definition 8. *For a given test* t *a* test annotation *is a function*

$$a : ctraces\,(t) \longrightarrow \{pass, fail\}.$$

A pair $\widehat{t} = (t, a)$ *consisting of a test and a test annotation is called an annotated test. The set of all such* $\widehat{t}$*, denoted by* $\widehat{T} = \left\{(t_i, a_i)_{i\in\mathcal{I}}\right\}$ *for some index set* $\mathcal{I}$*, is called an* annotated test suite*. If* t *is a test case for a specification* $\mathcal{A}_{\mathcal{S}}$ *we define the test annotation* $a_{\mathcal{A}_{\mathcal{S}},t} : ctraces\,(t) \longrightarrow \{pass, fail\}$ *by*

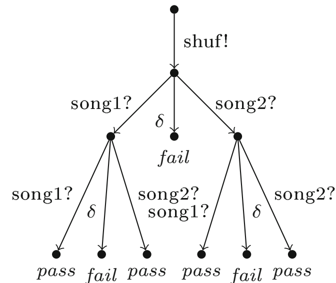
$$a_{\mathcal{A}_{\mathcal{S}},t}\,(\sigma) = \begin{cases} fail & if\, \exists \varrho \in traces\,(\mathcal{A}_{\mathcal{S}})\,, a! \in L_O^{\delta} : \varrho a! \sqsubseteq \sigma \wedge \varrho a! \notin traces\,(\mathcal{A}_{\mathcal{S}})\,; \\ pass & otherwise. \end{cases}$$

Example 9. Figure 2 shows two derived tests for the specification in Fig. 1. Note that the action signature is mirrored. Therefore if $s \xrightarrow{\mu,a}_{\widehat{t}} s'$ with a an output action of the specification, then we have $\mu = Dirac$. Test $\widehat{t_2}$ shows how we apply stimuli, observe or stop with probabilities $\frac{1}{3}$ each. If we stimulate, we apply stop! and shuf! with probability $\frac{1}{2}$ each.

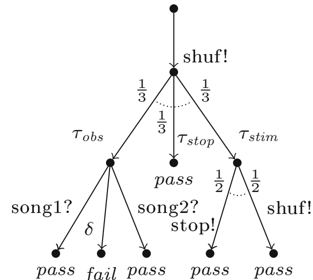
Algorithms. The procedure *batch* in Algorithm 1 generates test cases from a specification, given a specification pIOTSs $\mathcal{A}_{\mathcal{S}}$ and a history σ, which is initially ϵ. Each step a probabilistic choice is made to return an empty test, to observe or to stimulate, denoted with probabilities $p_{\sigma,1}, p_{\sigma,2}$ or $p_{\sigma,3}$ respectively. The latter two call the procedure *batch* again. If erroneous output is detected, we stop immediately. We require that $p_{\sigma,1} + p_{\sigma,2} + p_{\sigma,3} = 1$.

Algorithm 2 shows a sound way to derive tests on-the-fly. The inputs are a specification $\mathcal{A}_{\mathcal{S}}$, a concrete implementation $\mathcal{A}_{\mathcal{I}}$ and a test length $n \in \mathbb{N}$. The algorithm returns a verdict of whether or not the implementation is *ioco* correct in the first n steps. If erroneous output was detected, the verdict will be *fail* and *pass* otherwise. With probability $p_{\sigma,1}$ we observe and with probability $p_{\sigma,2}$ we stimulate. The algorithm stops after n steps. Thus, $p_{\sigma,1} + p_{\sigma,2} = 1$.

Theorem 10. *All test cases generated by Algorithm 1 are test cases according to Definition 7. All test cases generated by Algorithm 2 assign the correct verdict according to Definition 8.*



(a) Annotated test $\hat{t}_1$



(b) Prob. annotated test $\hat{t}_2$

Fig. 2. Two tests derived from the specification in Fig. 1

Algorithm 1: Batch test generation for *pioco*.

```
    Input: Specification pIOTS A_S and
           history σ ∈ traces(A_S).
    Output: A test case t for A_S.
 1  Procedure batch(A_S, σ)
 2      p_{σ,1}·[true] →
 3      return {τ_stop}
 4      p_{σ,2}·[true] →
 5      result := {τ_obs}
 6      forall b! ∈ L_O do:
 7          if σb! ∈ traces(A_S) :
 8              result := result ∪ {b!σ' |
                σ' ∈ batch(A_S, σb!)}
 9          else:
10              result := result ∪ {b!}
11          end
12      end
13      return result
14      p_{σ,3}·[σa? ∈ traces(A_S)] →
15      result := {τ_stim} ∪
        {a?σ' | σ' ∈ batch(A_S, σa?)}
16      forall b! ∈ L_O do:
17          if σb! ∈ traces(A_S) :
18              result := result ∪ {b!σ' |
                σ' ∈ batch(A_S, σb!)}
19          else:
20              result := result ∪ {b!}
21          end
22      end
23      return result
```

Algorithm 2: On-the-fly test case derivation for *pioco*.

```
    Input: Specification pIOTS A_S,
           an implementation A_I and
           an upper bound for the
           test length n ∈ ℕ.
    Output: Verdict pass if Impl. was
            ioco conform in the first
            n steps and fail if not.
 1  σ := ε
 2  while |σ| < n do:
 3      p_{σ,1}·[true] →
 4      observe next output b!
        (possibly δ) of A_I
 5      σ := σb!
 6      if σ ∉ traces(A_S) :
 7          return fail
 8      p_{σ,2} · [σa? ∈ traces(A_S)] →
 9      try:
10          atomic
11              stimulate I with a?
12              σ := σa?
13          end
14      catch an output b! occurs
        before a? could be applied
15          σ := σb!
16          if σ ∉ traces(A_S) :
17              return fail
18
19      end
20  end
21  return pass
```

3.2 Test Evaluation

In our framework, we assess functional behaviour by the test verdict $a_{\mathcal{A}_{\mathcal{S}},t}$ and probabilistic behaviour via statistics, as elaborated below.

Statistical Verdict. Given a (black box) implementation, the idea is to run an offline or online test case multiple times, in order to collect a sample. Then, we check if the frequencies of the traces contained in this sample match the probabilities in the specification via statistical hypothesis testing. However, since the specification contains non-determism, we cannot apply statistical means directly. Rather, we check if the observed trace frequencies can be explained, if we resolve occurring non-determinism in the specification according to some scheduler.

We formulate a hypothesised scheduler that makes the occurrence of the sample most likely. This gives rise to a purely probabilistic computation tree and probabilities and expected values for each trace can be calculated. Based on a predefined level of significance $\alpha \in (0, 1)$ we use null hypothesis testing to determine whether to accept or reject the hypothesised scheduler. If it is accepted, we have no reason to assume that the implementation differs probabilistically from the specification and give the *pass* label. If it is rejected, we assign the *fail* verdict, because there is no scheduler to explain the observed frequencies.

Sampling. To collect a sample, we define the length $k \in \mathbb{N}$ and width $m \in \mathbb{N}$ of an experiment first, i.e. how long shall we observe the machine and how many times do we want to run it before stopping. Thus, we collect $\sigma_1, \ldots, \sigma_m \in \mathit{traces}\,(\mathcal{A}_\mathcal{I})$ with $|\sigma_i| = k$ for $i = 1, \ldots, m$. We call $O = (\sigma_1, \ldots, \sigma_m) \in \left(L^k\right)^m$ a *sample.* We assume the system is governed by a trace distribution D_i in every run, thus running the machine m times, means that a sample is generated by a sequence of m (possibly) different trace distributions $\boldsymbol{D} = (D_1, D_2, \ldots, D_m) \in \mathit{trd}\,(\mathcal{A}_\mathcal{I}, k)^m$.

Each run the implementation makes two choices. (1) It chooses a trace distribution D_i and (2) D_i chooses a trace σ_i. Once a trace distribution D_i is chosen, it is solely responsible for the trace σ_i, meaning that for $i \neq j$ the choice of σ_i by D_i is independent of the choice σ_j by D_j.

Frequencies. The frequency function is defined as $\mathit{freq} : \left(L^k\right)^m \to \mathit{Distr}\left(L^k\right)$, such that $\mathit{freq}\,(O)\,(\sigma) = \frac{|\{i=1,\ldots,m \wedge \sigma=\sigma_i\}|}{m}$. Assume that $k, m \in \mathbb{N}$, $\boldsymbol{D}$ and $\sigma \in L^k$ are fixed. Then a sample O can be treated as a Bernoulli experiment of length m, where success occurs in position $i \in \{1, \ldots, m\}$ if $\sigma = \sigma_i$. Thus, the success probability in the i-th step is given by $P_{D_i}\,(\sigma)$. So assume X_i are Bernoulli distributed random variables for $i = 1, \ldots, m$. We define a new random variable as $Z = \frac{1}{m}\sum_{i=1}^m X_i$, which represents the frequency of success in m steps governed by $\boldsymbol{D}$. Thus the expected frequency is given as

$$\mathbb{E}_\sigma^{\boldsymbol{D}} := \mathbb{E}\,(Z) = \frac{1}{m}\sum_{i=1}^m \mathbb{E}\,(X_i) = \frac{1}{m}\sum_{i=1}^m P_{D_i}\,(\sigma).$$

It is $\sum_\sigma \mathbb{E}_\sigma^{\boldsymbol{D}} = 1$, which means $\mathbb{E}^{\boldsymbol{D}}$ is the distribution expected under $\boldsymbol{D}$.

Acceptable Outcomes. We will accept a sample O if $\mathit{freq}\,(O)$ lies within some distance r of the expected distribution $\mathbb{E}^{\boldsymbol{D}}$. Recall the definition of a ball centred at $x \in X$ with radius r as $B_r\,(x) = \{y \in X \mid \mathit{dist}\,(x, y) \leq r\}$. All distributions deviating at most by r from the expected distribution are contained within the ball $B_r\left(\mathbb{E}^{\boldsymbol{D}}\right)$, where $\mathit{dist}\,(u, v) := \sup_{\sigma \in L^k} \mid u\,(\sigma) - v\,(\sigma) \mid$ and u and v are distributions. In order to minimize the error of falsely accepting a sample, we choose the smallest radius, such that the error of falsely rejecting a sample is not greater than a predefined level of significance $\alpha \in (0, 1)$ by $\bar{r} := \inf\left\{r \mid P_{\boldsymbol{D}}\left(\mathit{freq}^{-1}\left(B_r\left(\mathbb{E}^{\boldsymbol{D}}\right)\right)\right) > 1 - \alpha\right\}$.

Definition 11. *For $k, m \in \mathbb{N}$ and a pIOTS $\mathcal{A}$ the* acceptable outcomes *under $\boldsymbol{D} \in \mathit{trd}\,(\mathcal{A}, k)^m$ of significance level $\alpha \in (0, 1)$ are given by the set of observations $\mathit{Obs}\,(\boldsymbol{D}, \alpha, k, m) = \left\{O \in \left(L^k\right)^m \mid \mathit{dist}\left(\mathit{freq}\,(O), \mathbb{E}^{\boldsymbol{D}}\right) \leq \bar{r}\right\}$. The set of* observations *of $\mathcal{A}$ of significance level $\alpha \in (0, 1)$ is given by*

$$\mathit{Obs}\,(\mathcal{A}, \alpha, k, m) = \bigcup_{\boldsymbol{D} \in \mathit{trd}(\mathcal{A}, k)^m} \mathit{Obs}\,(\boldsymbol{D}, \alpha, k, m).$$

The defined set of observations of a pIOTS $\mathcal{A}$ therefore has two properties, reflecting the error of false rejection and false acceptance respectively.

1. For $\boldsymbol{D} \in trd\,(\mathcal{A})$ of length k, we have $P_{\boldsymbol{D}}\,(Obs\,(\mathcal{A}, \alpha, k, m)) \geq 1 - \alpha$,
2. For $\boldsymbol{D}' \notin trd\,(\mathcal{A})$ of length k, we have $P_{\boldsymbol{D}'}\,(Obs\,(\mathcal{A}, \alpha, k, m)) \leq \beta_m$,

where α is the predefined level of significance and β_m is unknown but minimal by construction. Note that $\beta_m \to 0$ as $m \to \infty$, thus the error of falsely accepting an observation decreases with increasing sample width.

Application. This framework has two problems for practical applications: (1) the parameter $\bar{r}$ may be hard to find and (2) for a given sample, it is no trivial task to find the trace distribution, that gives it maximal likelihood, i.e.

$$\mathbb{P}^{k,m}_{\mathcal{A}}\,(O) := \max_{\boldsymbol{D} \in (trd(\mathcal{A},k) \setminus trd(\mathcal{A},k-1))^m} P_{\boldsymbol{D}}\,(O).$$

The parameter $\bar{r}$ gives the best fit, but finding it is no trivial task. It is of interest for the soundness and completeness proofs, but in practice we will use χ^2 hypothesis testing. The empirical value $\chi^2 = \sum_{i=1}^{m} (n\,(\sigma_i) - mE^{\boldsymbol{D}}_{\sigma_i})^2 / mE^{\boldsymbol{D}}_{\sigma_i}$, where $n\,(\sigma)$ is the amount σ occurred in the sample, is compared to critical values of given degrees of freedom and levels of significance. These values can be calculated or looked up in a χ^2 table.

Since expectations in our construction depend on a scheduler/trace distribution to explain a possible sample, it is of interest to find the best fit. Hence, we are trying to solve the minimisation

$$\min_{\boldsymbol{D}} \sum_{i=1}^{m} \frac{\left(n\,(\sigma_i) - mE^{\boldsymbol{D}}_{\sigma_i}\right)^2}{mE^{\boldsymbol{D}}_{\sigma_i}}. \tag{1}$$

By construction, we want to optimize the probabilities p_i used by a scheduler to resolve non-determinism. This turns (1) into a minimisation of a rational function $f\,(p)\,/g\,(p)$ with inequality constraints on the vector p. As shown in [25], minimizing rational functions is **NP**-hard. This approach optimizes one possible trace distribution to fit the sample data instead of finding m different ones. This topic could be handled in future research, with the assumption of one distribution which lets the implementation choose different trace distributions.

Verdict Function. With this framework, the following decision process summarizes if an implementation fails for functional and/or statistical behaviour.

Definition 12. *Given a specification $\mathcal{A}_{\mathcal{S}}$, an annotated test $\widehat{t}$ for $\mathcal{A}_{\mathcal{S}}$, $k, m \in \mathbb{N}$ where k given by the trace length of $\widehat{t}$ and a level of significance $\alpha \in (0,1)$, we define the* functional verdict *as the function $v_{\widehat{t}} : pIOTS \longrightarrow \{pass, fail\}$, with*

$$v_{\widehat{t}}\,(\mathcal{A}_{\mathcal{I}}) = \begin{cases} pass & if\ \forall \sigma \in ctraces\,(\mathcal{A}_{\mathcal{I}} \,||\, t) \cap ctraces\,(t) : a\,(\sigma) = pass \\ fail & otherwise, \end{cases}$$

the statistical verdict *as the function $v^{\alpha,m}_{t} : pIOTS \longrightarrow \{pass, fail\}$, with*

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

and finally the overall verdict *as the function* $V_{\hat{t}}^{\alpha,m} : pIOTS \to \{pass, fail\}$, *with* $V_{\hat{t}}^{\alpha,m}(\mathcal{A}_{\mathcal{I}}) = pass$ *if* $v_{\hat{t}}(\mathcal{A}_{\mathcal{I}}) = v_{t}^{\alpha,m}(\mathcal{A}_{\mathcal{I}}) = pass$ *and* $V_{\hat{t}}^{\alpha,m}(\mathcal{A}_{\mathcal{I}}) = fail$ *otherwise. For an annotated test suite* $\widehat{T}$ *for* $\mathcal{A}_{\mathcal{S}}$ *we lift this to* $V_{\widehat{T}}^{\alpha,m}(\mathcal{A}_{\mathcal{I}}) = pass$ *if* $V_{\hat{t}}^{\alpha,m}(\mathcal{A}_{\mathcal{I}}) = pass$ *for each* $\hat{t} \in \widehat{T}$ *and* $V_{\widehat{T}}^{\alpha,m}(\mathcal{A}_{\mathcal{I}}) = fail$ *otherwise.*

4 Conformance, Soundness and Completeness

A key result of our paper is the correctness of our framework, formalized as soundness and completeness. *Soundness* states that each test case is assigned the correct verdict. *Completeness* states that the framework is powerful enough to discover each deviation from the specification. Formulating these properties requires a formal notion of conformance that we formalize as the *pioco*-relation.

4.1 Probabilistic Input/Output Conformance $\sqsubseteq_{pioco}$

The classical *ioco* relation [40] states that an implementation conforms to a specification, if it never provides any unspecified output or quiescence, i.e. for two IOTSs $\mathcal{A}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{S}}$, with $\mathcal{A}_{\mathcal{I}}$ input-enabled, we say $\mathcal{A}_{\mathcal{I}} \sqsubseteq_{ioco} \mathcal{A}_{\mathcal{S}}$, iff

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

To generalize *ioco* to pIOTSs, we introduce two auxiliary concepts:

1. the prefix relation for trace distributions $H \sqsubseteq_k H'$ is the analogue of trace prefixes, i.e. $H \sqsubseteq_k H'$ iff $\forall \sigma \in L^k : P_H(\sigma) = P_{H'}(\sigma)$
2. for a pIOTSs $\mathcal{A}$ and a trace distribution H of length k, the *output continuation* of H in $\mathcal{A}$ contains all trace distributions, which are equal up to length k and assign every trace of length $k+1$ ending in input probability 0. We set

$$outcont(H, \mathcal{A}) := \left\{ H' \in trd(\mathcal{A}, k+1) \mid H \sqsubseteq_k H' \wedge \forall \sigma \in L^k L_I : P_{H'}(\sigma) = 0 \right\}.$$

Intuitively an implementation should conform to a specification, if the probability of every trace in $\mathcal{A}_{\mathcal{I}}$ specified in $\mathcal{A}_{\mathcal{S}}$, can be matched. Just like in *ioco*, we neglect unspecified traces ending in input actions. However, if there is unspecified output in the implementation, there is at least one adversary that schedules positive probability to this continuation.

Definition 13. *Let* $\mathcal{A}_{\mathcal{I}}$ *and* $\mathcal{A}_{\mathcal{S}}$ *be two pIOTSs. Furthermore let* $\mathcal{A}_{\mathcal{I}}$ *be input-enabled, then we say* $\mathcal{A}_{\mathcal{I}} \sqsubseteq_{pioco} \mathcal{A}_{\mathcal{S}}$ *iff*

$$\forall k \in \mathbb{N} \forall H \in trd(\mathcal{A}_{\mathcal{S}}, k) : outcont(H, \mathcal{A}_{\mathcal{I}}) \subseteq outcont(H, \mathcal{A}_{\mathcal{S}}).$$

The *pioco* relation conservatively extends the *ioco* relation, i.e. both relations coincide for IOTSs.

Theorem 14. *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be two IOTSs and* $\mathcal{A}$ *be input-enabled, then*

$$\mathcal{A} \sqsubseteq_{ioco} \mathcal{B} \Longleftrightarrow \mathcal{A} \sqsubseteq_{pioco} \mathcal{B}.$$

The implementation is always assumed to be input-enabled. If the specification is input-enabled too, then *pioco* coincides with trace distribution inclusion. Moreover, our results show that *pioco* is transitive, just like *ioco*.

Theorem 15. *Let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be pIOTSs and let $\mathcal{A}$ and $\mathcal{B}$ be input-enabled, then*

- $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$ *if and only if* $\mathcal{A} \sqsubseteq_{TD} \mathcal{B}$.
- $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$ *and* $\mathcal{B} \sqsubseteq_{pioco} \mathcal{C}$ *then* $\mathcal{A} \sqsubseteq_{pioco} \mathcal{C}$.

4.2 Soundness and Completeness

Talking about soundness and completeness when referring to probabilistic systems is not a trivial topic, since one of the main inherent difficulties of statistical analysis is the possibility of false rejection or false acceptance.

The former is of interest when we refer to soundness (i.e. what is the probability that we erroneously assign *fail* to a correct implementation), and the latter is important when we talk about completeness (i.e. what is the probability that we assign *pass* to an erroneous implementation). Thus, a test suite can only fulfil these properties with a guaranteed (high) probability (c.f. Definition 12).

Definition 16. *Let $\mathcal{A}_\mathcal{S}$ be a specification over an action signature (L_I, L_O), $\alpha \in (0,1)$ the level of significance and $\widehat{T}$ an annotated test suite for $\mathcal{A}_\mathcal{S}$. Then*

- $\widehat{T}$ *is sound for $\mathcal{A}_\mathcal{S}$ with respect to $\sqsubseteq_{pioco}$, if for all input-enabled implementations $\mathcal{A}_i \in pIOTS$ and sufficiently large $m \in \mathbb{N}$ it holds that*

$$\mathcal{A}_\mathcal{I} \sqsubseteq_{pioco} \mathcal{A}_\mathcal{S} \Longrightarrow V_{\widehat{T}}^{\alpha,m}(\mathcal{A}_\mathcal{I}) = pass.$$

- $\widehat{T}$ *is complete for $\mathcal{A}_\mathcal{S}$ with respect to $\sqsubseteq_{pioco}$, if for all input-enabled implementations $\mathcal{A}_\mathcal{I} \in pIOTS$ and sufficiently large $m \in \mathbb{N}$ it holds that*

$$\mathcal{A}_\mathcal{I} \not\sqsubseteq_{pioco} \mathcal{A}_\mathcal{S} \Longrightarrow V_{\widehat{T}}^{\alpha,m}(\mathcal{A}_\mathcal{I}) = fail.$$

Soundness for a given $\alpha \in (0,1)$ expresses that we have a $1-\alpha$ chance that a correct system will pass the annotated suite for sufficiently large sample width m. This relates to false rejection of a correct hypothesis or correct implementation respectively.

Theorem 17 *(Soundness). Each annotated test for a pIOTS $\mathcal{A}_\mathcal{S}$ is sound for every level of significance $\alpha \in (0,1)$ wrt pioco.*

Completeness of a test suite is inherently a theoretic result. Since we allow loops, we require a test suite of infinite size. Moreover, there is still the chance of falsely accepting an erroneous implementation. However, this is bound from above by construction, and will decrease for bigger sample sizes (c.f. Definition 11).

Theorem 18 *(Completeness). The set of all annotated test cases for a specification $\mathcal{A}_\mathcal{S}$ is complete for every level of significance $\alpha \in (0,1)$ wrt pioco.*

5 Experimental Validation

To apply our framework, we implemented two well-known randomized communication protocols in Java, and tested these with the MBT tool JTorX [3]. The statistical verdicts were calculated in MatLab with a level of significance $\alpha = 0.1$.

5.1 Binary Exponential Backoff

The Binary Exponential Backoff protocol is a data transmission protocol between N hosts, trying to send information via one bus [19]. If two hosts send simultaneously, then their messages collide and they pick a new waiting time before trying again: after i collisions, they randomly choose a slot in $\{0, \ldots 2^i - 1\}$ until the message gets through.

A sample of the protocol is shown in Table 1. Note that our specification of this protocol contains no non-determinism. Thus, calculations in this example are not subject to optimization to find the best trace distribution.

Table 1. A sample O of trace length $k = 5$ and depth (number of test runs) $m = 10^5$. Calculations yield $\chi^2 = 14.84 < 17.28 = \chi^2_{0.1}$, hence we accept the implementation.

#	Trace σ	n	$\approx mE_\sigma$	$[l_{0.1}, u_{0.1}]$	$\approx \frac{(n-mE_\sigma)^2}{mE_\sigma}$
1	collide! send! collide! send! send!	18656	18750	$[18592, 18907]$	0.47
2	collide! send! collide! send! collide!	18608	18750	$[18592, 18907]$	1.08
3	collide! collide! send! collide! send!	16473	16408	$[16258, 16557]$	0.26
4	collide! collide! send! send! collide!	12665	12500	$[12366, 12633]$	2.18
5	collide! send! collide! collide! send!	11096	10938	$[10811, 11064]$	2.28
6	collide! collide! collide! send! send!	8231	8203	$[8091, 8314]$	0.10
7	collide! collide! send! send! send!	6108	6250	$[6152, 6347]$	3.23
8	collide! collide! collide! send! collide!	2813	2734	$[2667, 2800]$	2.28
9	collide! collide! send! collide! collide!	2291	2344	$[2282, 2405]$	1.20
10	collide! send! collide! collide! collide!	1538	1563	$[1512, 1613]$	0.40
11	collide! collide! collide! collide! send!	1421	1465	$[1416, 1513]$	1.32
12	collide! collide! collide! collide! collide!	100	98	$[85, 110]$	0.04
				$\chi^2 =$	14.84
				Verdict:	*Accept*

n in Table 1 shows how many times each trace occurred and E_σ gives the expected value. The interval $[l_{0.1}, r_{0.1}]$ represents the 90 % confidence interval under the assumption of a normal distribution. It gives a rough idea how much values will deviate for the given level of confidence. However, we are interested in the multinomial deviation (i.e. less deviation of one trace allows higher deviation for another trace). For that purpose we use the χ^2 score, given by the sum of the entries of the last column. Calculation shows $\chi^2 = 14.84 < 17.28 = \chi^2_{0.1}$, which is the critical value for 11 degrees of freedom and $\alpha = 0.1$. Consequently, we accept the hypothesis of the probabilities being implemented correctly.

5.2 IEEE 1394 FireWire Root Contention Protocol

The IEEE 1394 FireWire Root Contention Protocol [37] elects a leader between two nodes via coin flips: If *head* comes up, node i picks a waiting time $fast_i \in [0.24\,\mu s, 0.26\,\mu s]$, if *tail* comes up, it waits $slow_i \in [0.57\,\mu s, 0.60\,\mu s]$. After the waiting time has elapsed, the node checks whether a message has arrived: if so, the node declares itself leader. If not, the node will send out a message itself, asking the other node to be the leader. Thus, the four outcomes of the coin flips are: $\{fast_1, fast_2\}, \{slow_1, slow_2\}, \{fast_1, slow_2\}$ and $\{slow_1, fast_2\}$. The protocol contains inherent non-determinism [37]; If different times were picked, the protocol always terminates. However, if equal times were picked, it may either elect a leader, or retry depending on the resolution of the non-determinism.

Table 2. A sample O of length $k = 5$ and depth $m = 10^5$ of the FireWire root contention protocol. Calculations of χ^2 are done after optimization in p.

#	Trace σ	$\approx mE_\sigma^D(p)$	*Correct* n_c	M_1 n_{M_1}	M_2 n_{M_2}	M_3 n_{M_3}	M_4 n_{M_4}
1	c1? slow$_1$! c2? slow$_2$! retry!	$6250 \cdot p$	3148	1113	3091	3055	3161
2	c1? slow$_1$! c2? slow$_2$! done!	$18750 \cdot p$	9393	3361	9047	9242	9329
3	c1? slow$_1$! c2! fast$_2$! done!	$25000 \cdot p$	12531	40507	18163	15129	12982
4	c1? fast$_1$! c2! fast$_2$! retry!	$8333 \cdot p$	4254	1467	4037	4066	4179
5	c1? fast$_1$! c2! fast$_2$! done!	$16667 \cdot p$	8227	3048	7858	8474	8444
6	c1? fast$_1$! c2? slow$_2$! done!	$25000 \cdot p$	12438	504	7918	10128	11867
7	c2? slow$_2$! c1? slow$_1$! retry!	$6250 \cdot (1-p)$	3073	1137	2961	3256	3135
8	c2? slow$_2$! c1? slow$_1$! done!	$18750 \cdot (1-p)$	9231	3427	9069	9456	9368
9	c2? slow$_2$! c1? fast$_1$! done!	$25000 \cdot (1-p)$	12657	447	8055	9685	11975
10	c2? fast$_2$! c1? fast$_1$! retry!	$8333 \cdot (1-p)$	4211	1466	4008	4131	4199
11	c2? fast$_2$! c1? fast$_1$! done!	$16667 \cdot (1-p)$	8335	2977	7969	8295	8312
12	c2? fast$_2$! c1? slow$_1$! done!	$25000 \cdot (1-p)$	12502	40546	17824	15083	13049
		$p_{opt} \approx$	0.499	0.498	0.502	0.500	0.499
		$\chi^2 \approx$	9.34	169300	8175	2185	99.22
		Verdict	*Accept*	*Reject*	*Reject*	*Reject*	*Reject*

Table 2 shows the recorded traces, where c1? and c2? denote coin1 and coin2 respectively. We have tested five implementations: Implementation *Correct* implements fair coins, while the mutants M_1, M_2, M_3 and M_4 were subjects to probabilistic deviations giving advantage to the second node, i.e. $P(fast_1) = P(slow_2) = 0.1$, $P(fast_1) = P(slow_2) = 0.4$, $P(fast_1) = P(slow_2) = 0.45$ and $P(fast_1) = P(slow_2) = 0.49$ for mutants 1, 2, 3 and 4 respectively. The expected value $E_\sigma^{\boldsymbol{D}}$ depends on resolving one non-determinism by varying p (which coin was flipped first). Note that other non-determinism was not subject to optimization, but immediately clear by trace frequencies. The calculated χ^2 scores are based on an optimized value for p for each sample and compared to the critical value $\chi^2_{0.1} = 17.28$ resulting in the verdicts shown.

6 Conclusions and Future Work

We defined a sound and complete framework to test probabilistic systems, defined a conformance relation in the *ioco* tradition called *pioco* and showed how to derive probabilistic tests of a requirements model. Verdicts that handle the functional and statistical behaviour are assigned after a test is applied. We showed that the correct verdict can be assigned up to arbitrary precision by setting a level of significance and sufficiently large sample size.

Future work should focus on the practical aspects of our theory: tool support, larger case studies and more powerful statistical methods to increase efficiency.

References

1. Al-Karaki, J.N., Kamal, A.E.: Routing techniques in wireless sensor networks: a survey. IEEE Wireless Commun. **11**(6), 6–28 (2004)
2. Beaumont, M.A., Zhang, W., Balding, D.J.: Approximate bayesian computation in population genetics. Genetics **162**(4), 2025–2035 (2002)
3. Belinfante, A.: JTorX: a tool for on-line model-driven test derivation and execution. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 266–270. Springer, Heidelberg (2010)
4. Beyer, M., Dulz, W.: Scenario-based statistical testing of quality of service requirements. In: Leue, S., Systä, T.J. (eds.) Scenarios: Models, Transformations and Tools. LNCS, vol. 3466, pp. 152–173. Springer, Heidelberg (2005)
5. Bohnenkamp, H.C., Belinfante, A.: Timed testing with TorX. In: Fitzgerald, J.S., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 173–188. Springer, Heidelberg (2005)
6. Böhr, F.: Model based statistical testing of embedded systems. In: IEEE 4th International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 18–25 (2011)
7. Bozga, M., David, A., Hartmanns, A., Hermanns, H., Larsen, K.G., Legay, A., Tretmans, J.: State-of-the-art tools and techniques for quantitative modeling and analysis of embedded systems. In: DATE, pp. 370–375 (2012)
8. Briones, Laura Brandán, Brinksma, Ed: A test generation framework for *quiescent* real-time systems. In: Grabowski, Jens, Nielsen, Brian (eds.) FATES 2004. LNCS, vol. 3395, pp. 64–78. Springer, Heidelberg (2005)
9. Cheung, L., Stoelinga, M., Vaandrager, F.: A testing scenario for probabilistic automata. J. ACM **54**(6), 45 (2007). Article No. 29
10. Choi, S.G., Dachman-Soled, D., Malkin, T., Wee, H.: Improved non-committing encryption with applications to adaptively secure protocols. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 287–302. Springer, Heidelberg (2009)
11. Cleaveland, R., Dayar, Z., Smolka, S.A., Yuen, S.: Testing preorders for probabilistic processes. Inform. Comput. **154**(2), 93–148 (1999)
12. Cohn, D.L.: Measure Theory. Birkhäuser, Boston (1980)
13. Deng, Y., Hennessy, M., van Glabbeek, R.J., Morgan, C.: Characterising Testing Preorders for Finite Probabilistic Processes. CoRR (2008)
14. Gerhold, M., Stoelinga, M.: Ioco Theory for Probabilistic Automata. In: Proceedings of the Tenth Workshop on MBT, pp. 23–40 (2015)

15. van Glabbeek, R.J., Smolka, S.A., Steffen, B., Tofts, C.: Reactive, Generative, and Stratified Models of Probabilistic Processes, pp. 130–141. IEEE Computer Society Press (1990)
16. Hierons, R.M., Núñez, M.: Testing probabilistic distributed systems. In: Hatcliff, J., Zucca, E. (eds.) FMOODS 2010, Part II. LNCS, vol. 6117, pp. 63–77. Springer, Heidelberg (2010)
17. Hierons, R.M., Merayo, M.G.: Mutation testing from probabilistic and stochastic finite state machines. J. Syst. Softw. **82**, 1804–1818 (2009)
18. Hwang, I., Cavalli, A.R.: Testing a probabilistic FSM using interval estimation. Comput. Netw. **54**, 1108–1125 (2010)
19. Jeannet, B., D'Argenio, P.R., Larsen, K.G.: Rapture: a tool for verifying markov decision processes. In: Tools Day (2002)
20. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 200–204. Springer, Heidelberg (2002)
21. Larsen, K.G., Skou, A.: Bisimulation Through Probabilistic Testing, pp. 344–352. ACM Press (1989)
22. Larsen, K.G., Mikucionis, M., Nielsen, B.: Online testing of real-time systems using UPPAAL. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, pp. 79–94. Springer, Heidelberg (2005)
23. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley, New York (1994)
24. Merayo, M.G., Hwang, I., Núñez, M., Cavalli, A.: A statistical approach to test stochastic and probabilistic systems. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 186–205. Springer, Heidelberg (2009)
25. Nie, J., Demmel, J., Gu, M.: Global minimization of rational functions and the nearest GCDs. J. Global Optim. **40**(4), 697–718 (2008)
26. Paige, B., Wood, F.: A Compilation Target for Probabilistic Programming Languages. CoRR arXiv:1403.0504 (2014)
27. Peters, H., Knieke, C., Brox, O., Jauns-Seyfried, S., Krämer, M., Schulze, A.: A test-driven approach for model-based development of powertrain functions. In: Cantone, G., Marchesi, M. (eds.) XP 2014. LNBIP, vol. 179, pp. 294–301. Springer, Heidelberg (2014)
28. Pfeffer, A.: Practical probabilistic programming. In: Frasconi, P., Lisi, F.A. (eds.) ILP 2010. LNCS, vol. 6489, pp. 2–3. Springer, Heidelberg (2011)
29. Prowell, S.J.: Computations for Markov Chain Usage Models. Technical Report (2003)
30. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (2014)
31. Remke, A., Stoelinga, M. (eds.): Stochastic Model Checking. LNCS, vol. 8453. Springer, Heidelberg (2014)
32. Russell, N., Moore, R.: Explicit modelling of state occupancy in hidden markov models for automatic speech recognition. In: Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP 1985, vol. 10, pp. 5–8 (1985)
33. Segala, R.: Modeling and verification of randomized distributed real-time systems. Ph.D. thesis, Cambridge, MA, USA (1995)
34. Segala, R.: Testing Probabilistic Automata. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 299–314. Springer, Heidelberg (1996)
35. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 202–215. Springer, Heidelberg (2004)

36. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 266–280. Springer, Heidelberg (2005)
37. Stoelinga, M., Vaandrager, F.W.: Root contention in IEEE 1394. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 53–74. Springer, Heidelberg (1999)
38. Stoelinga, M.: Alea jacta est: verification of probabilistic, real-time and parametric systems. Ph.D. thesis, Radboud University of Nijmegen (2002)
39. Stokkink, W.G.J., Timmer, M., Stoelinga, M.I.A.: Divergent quiescent transition systems. In: Veanes, M., Viganò, L. (eds.) TAP 2013. LNCS, vol. 7942, pp. 214–231. Springer, Heidelberg (2013)
40. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Softw. Concepts Tools **17**(3), 103–120 (1996)
41. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008)
42. Walton, G.H., Poore, J.H., Trammell, C.J.: Statistical Testing of Software Based on a Usage Model. Softw. Pract. Exper. **25**(1), 97–108 (1995)
43. Whittaker, J.A., Thomason, M.G.: A markov chain model for statistical software testing. IEEE Trans. Softw. Eng. **20**(10), 812–824 (1994)