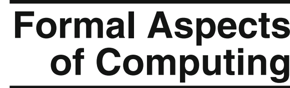
https://doi.org/10.1007/s00165-017-0440-4
The Author(s) © 2017. This article is an open access publication
Formal Aspects of Computing (2018) 30: 77–106



Model-based testing of probabilistic systems

Marcus Gerhold[1] and Mariëlle Stoelinga[1]

[1] Formal Methods and Tools Group, University of Twente, Enschede, The Netherlands

Abstract. This work presents an executable model-based testing framework for probabilistic systems with non-determinism. We provide algorithms to automatically generate, execute and evaluate test cases from a probabilistic requirements specification. The framework connects input/output conformance-theory with hypothesis testing: our algorithms handle functional correctness, while statistical methods assess, if the frequencies observed during the test process correspond to the probabilities specified in the requirements. At the core of our work lies the conformance relation for probabilistic input/output conformance, enabling us to pin down exactly when an implementation should pass a test case. We establish the correctness of our framework alongside this relation as soundness and completeness; Soundness states that a correct implementation indeed passes a test suite, while completeness states that the framework is powerful enough to discover each deviation from a specification up to arbitrary precision for a sufficiently large sample size. The underlying models are probabilistic automata that allow invisible internal progress. We incorporate divergent systems into our framework by phrasing four rules that each well-formed system needs to adhere to. This enables us to treat divergence as the absence of output, or quiescence, which is a well-studied formalism in model-based testing. Lastly, we illustrate the application of our framework on three case studies.

Keywords: Model-based testing; Probabilistic automaton; Trace distribution; Hypothesis testing

1. Introduction

Probability. Probability plays a crucial role in a vast number of computer applications. A large body of communication protocols and computation methods use randomized algorithms to achieve their goals. For instance, random walks are utilized in sensor networks [AK04], control policies in robotics lead to the emerging field of probabilistic robotics [TBF05], speech recognition makes use of hidden Markov models [RM85] and security protocols use random bits in their encryption methods [CDSMW09]. Such applications can be implemented in one of the many probabilistic programming languages, such as Probabilistic-C [PW14] or Figaro [Pfe11]. On a higher level, service level agreements are formulated in a stochastic fashions, for instance specifying that a certain up-time should be at least 99%.

Correspondence and offprint requests to: M. Gerhold and M. Stoelinga, E-mails: marcus.gerhold@gmail.com; m.gerhold@utwente.nl; m.i.a.stoelinga@utwente.nl

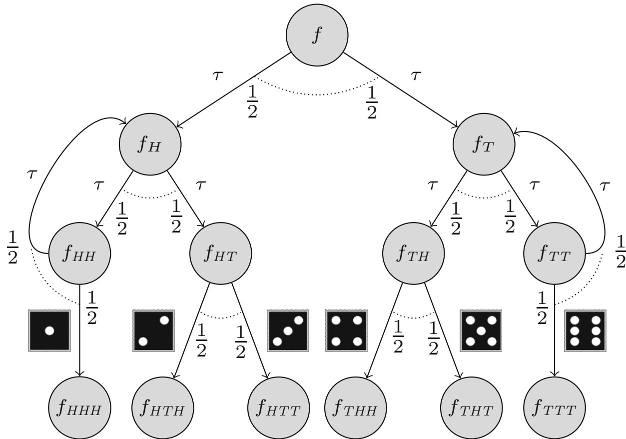


Fig. 1. Dice program based on Knuth and Yao [KY76]. A 6-sided die is simulated by repeated tosses of a fair coin

The key question is whether such probabilistic systems are correct: is bandwidth distributed fairly among all parties? Is the up-time and packet delay according to specification? Are security measures safe enough to withstand random attacks?

To investigate such questions, probabilistic verification has become a mature research field, putting forward models like probabilistic automata (PAs) [Seg95, Sto02], Markov decision processes [Put14], (generalized) stochastic Petri nets [MBC+94], and interactive Markov chains [Her02], with verification techniques like stochastic model checking [RS14], and supporting tools like Prism [KNP02], or Plasma [JLS12].

Testing. In practice however, testing is the most common validation technique. Testing of information and communication technology (ICT) systems is a vital process to establish their correctness. The system is subjected to many well-designed test cases that compare the outcome to a requirements specification. At the same time it is time consuming and costly, often taking up to 50% of all project resources [JS07]. Testing based on a model is a way to counteract this swiftly increasing demand.

Our work 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 system specification. It gained rapid popularity in industry by providing faster and more thorough means for the testing process, therefore lowering the overall costs in software development [JS07].

A wide variety of MBT frameworks exist, capable of handling different system aspects such as functional properties [Tre96], real-time [BB05, BB04, HLM+08], quantitative aspects [BDH+12], and continuous [PKB+14] and hybrid properties [vO06]. Surprisingly, there is only little work in the scientific community that focuses on executable testing frameworks for probabilistic systems, with notable exceptions being [HN10, HC10][1]. The presented work aims at filling this gap.

Probabilistic modelling. Our underlying models are a slight generalisations of the probabilistic automaton model [Seg95]. Figure 1 shows the dice simulation by Knuth and Yao [KY76]. In this application a fair 6-sided die is simulated by repeated coin tosses of a fair coin. Instead of moving from state to state, a transition moves from a state to a distribution over states. In this example, in the state f the model can go to the distribution over $\{f_H, f_T\}$ representing the outcomes of a coin toss *head* and *tail* with probability 0.5 each.

The PA model additionally facilitates non-deterministic choices. To illustrate, there might be a user dependent choice over whether to use a fair or unfair die in the simulation, as shown in Fig. 7. As argued in [Seg95] non-determinism is essential to model implementation freedom, interleaving and user behaviour. Probabilistic choices, on the other hand, model random choices made by the system, such as coin tosses, or by nature, such as degradation rates or failure probabilities. Having non-determinism in a model makes statistical analysis challenging, since an external observer does not know it is resolved.

[1] Note that the popular research branch of statistical testing, e.g., [BD05, WRT00], is concerned with choosing the test inputs probabilistically; it does not test for the correctness of the random choices made by the system itself.

One of the main challenges of our work consists of combining probabilistic choices and non-determinism in one test framework. As frequently done in literature [Seg95, Sto02], we resolve non-determinism via adversaries (a.k.a. policies or schedulers). In every step of the computation, an adversary decides for the system how to proceed. The resulting system can then be treated entirely probabilistically, since all non-deterministic choices were resolved. This enables us to do statistical analysis of the observable behaviour of the system under test (SUT).

Our contribution. The key results of our work are the soundness and completeness proofs of our framework. At their core lies a conformance relation, pinning down precisely what it means for an implementation to be considered *correct*. We choose the input/output conformance (ioco) relation known from the literature [Tre96, TBS11], since it is tailored to deal with non-determinism, and extend it with probabilities. The resulting relation is baptised probabilistic input-output conformance or *pioco*. Soundness states that a *pioco* correct implementation indeed passes a test suite. Albeit inherently a theoretical concept, completeness states that the framework is powerful enough to detect every faulty implementation.

We provide algorithms to automatically generate test cases from a requirements specification and execute them on the system under test (SUT). The verdicts, as part of the test case evaluation, can automatically be given after a sampling process and frequency analysis of observed traces.

The validity of our framework is illustrated with three case studies known from the literature exhibiting probabilistic behaviour: (1) the aforementioned dice application by Knuth and Yao [KY76], (2) the binary exponential backoff protocol [JDL02] and (3) the FireWire root contention protocol [SV99]. Our experimental set-up illustrates the use of possible tools and techniques to come to a conclusion about *pass* or *fail* verdicts of an implementation.

We show that, under certain constraints on the model, divergent behaviour, i.e. infinite invisible progress, can be treated as a special case of quiescence. Quiescence describes the indefinite absence of outputs in a system. Hence, an external observer can treat quiescence and divergence equivalently. We call a model adhering to these constraints *well-formed* and show that well-formedness is preserved under parallel composition. We provide means to transform a model into a well-formed one, thereby increasing the usage for practical modelling purposes. Thus, composing several subcomponents together still lets us apply our model-based testing methods.

The current version of this work presents an extension of [GS16]. We summarize the main novelties:

- fully fledged proofs of our results,
- additional examples and illustrations of our methods,
- support of invisible internal progress and divergent behaviour and
- a new case study.

Related work. Probabilistic testing preorders and equivalences are well studied [BB08, BNL13, CDSY99, DHvGM08, DLT08, HN17, Seg96], defining when two probabilistic transition systems are equivalent, or one subsumes the other. In particular, early and influential work is given by [LS89] and introduces the fundamental concepts of probabilistic bisimulation via hypothesis testing. Also, [CSV07] shows how to observe trace probabilities via hypothesis testing. Executable test frameworks for probabilistic systems have been defined for probabilistic finite state machines [HM09], dealing with mutations and stochastic timing, Petri nets [Böh11] and CSL [SVA04, SVA05].

The important research line of statistical testing [BD05, WPT95, WRT00] 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 of when to stop statistical testing is tackled in [Pro03].

An approach eminently similar to ours is by Hierons and Núñez [HN10, HN12]. However, our models can be considered as an extension of [HN10], reconciling probabilistic and non-deterministic choices in a fully fledged way. Being more restrictive enables [HN10, HN12] to focus on individual traces, whereas our approach uses trace distributions.

The current paper extends earlier work [GS15] that first introduced the *pioco* conformance relation and roughly sketched the test process. Extensions made later in [GS16] were (1) the more generic pIOTS model that includes invisible progress (a.k.a. internal actions), (2) the soundness and completeness results, (3) solid definitions of test cases, test execution, and verdicts, (4) the treatment of the absence of outputs (a.k.a. quiescence) and (5) the handling of probabilistic test cases. A later version [GS17] includes the aspect of stochastic time and extends our framework to the more general Markov automata.

Overview over the paper. [2] In Sect. 2 we establish the mathematical basics for our framework. Section 3 presents the automatic test generation and evaluation process alongside two algorithms. We experimentally validate our framework on three small case studies in Sect. 4. We present proofs that our method is sound and complete in Sect. 5. The inclusion of internal actions and possible resulting divergence in our systems is discussed in Sect. 6. Lastly, the paper ends with concluding remarks in Sect. 7.

2. Preliminaries

2.1. Probabilistic input/output systems

Probability theory. We assume the reader is acquainted with the basics of probability theory, but do recall integral definitions. In particular, we borrow the definition of probability spaces and their individual components rooted in measure theory. The interested reader is referred to [Coh80] for an excellent overview and further reading.

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 $\mathit{Distr}(X)$. The probability distribution that assigns probability 1 to a single element $x \in X$ is called the *Dirac* distribution over x and is denoted $\mathit{Dirac}(x)$.

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

Example 1 An intuitive illustration of a probability space is the one induced by a fair coin. If the coin is tossed, there is a 50% chance that it shows heads and 50% that it shows tails.

The sample space $\Omega = \{H, T\}$ contains these two outcomes. The event set $\mathcal{F} = \{\emptyset, \{H\}, \{T\}, \{H, T\}\}$ describes the possible events that may occur upon tossing the coin, i.e. (1) neither heads nor tails, (2) heads, (3) tails or (4) heads and tails. The probability measure that describes the intuitive understanding of a fair coin is then given as $\mathbb{P}(\emptyset) = 0$, $\mathbb{P}(\{H\}) = 0.5$, $\mathbb{P}(\{T\}) = 0.5$ and $\mathbb{P}(\{H, T\}) = 0$.

Hence, the triple $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space.

Probabilistic input/output systems. We introduce probabilistic input/output transition systems (pIOTSs) as an extension of labelled transition systems (LTSs) [TBS11, Tre08]. An LTS is a mathematical structure that models the behaviour of a system. It consists of states and edges between two states (a.k.a. transitions) labelled with action names. The states model the states the system can be in, whereas the labelled transitions model the actions that it can perform. Hence, we use 'label' and 'action' interchangeably.

Labelled transition systems are frequently modified to input/output systems by separating the action labels into distinct sets of input actions and output actions. Input actions are used to model the ways in which a user or the environment may interact with the system. The set of output actions represents the responses that a system can give. Occasionally, the system may advance internally without visibly making progress. This gives rise to the notion of internal or hidden actions.

In testing, a verdict must also be given if the implementation does not give any output at all [STS13]. To illustrate: If no input is provided to an ATM, it is certainly correct that no money is disbursed. However, having no money be output after a credit card and credentials are provided would be considered erroneous. We capture the absence of outputs (a.k.a. *quiescence*) with the special output action δ. This distinct label can be used to model that *no output* is desired in certain states.

We extend input/output transition systems with probabilities by having the target of transitions be distributions over states rather than a single state. Hence, if an action is executed in a state of the system, there is a probabilistic choice of which next state to go to next, cf. Fig. 2.

Following [GSST90], pIOTSs are defined as *input-reactive* and *output-generative*. Upon receiving an input, the pIOTS decides probabilistically which next state to move to. Upon producing an output, the pIOTS chooses both the output action and the state probabilistically. Mathematically, this means that each transition either involves one input action, or possibly several outputs, quiescence or internal actions. Note that a state can enable input and output transitions albeit not in the same distribution.

[2] Elaborate proofs of our results can be found in "appendix". We did not include them in the main text to maintain readability.

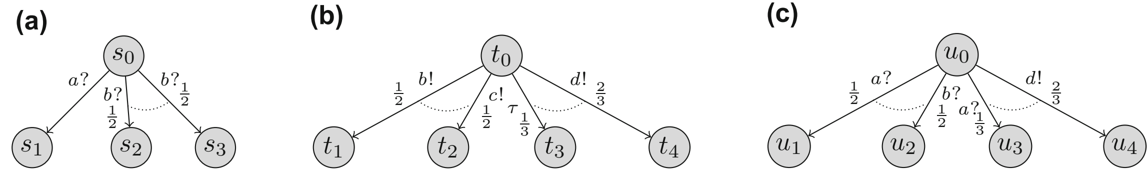


Fig. 2. Example models to illustrate *input-reactive* and *output-generative* transitions in pIOTSs. We use "?" to denote labels of the set of inputs and "!" to denote labels of the set of outputs. **a** Valid pIOTS, **b** valid pIOTS, **c** not a valid pIOTS

Definition 2 A *probabilistic input/output transition system* is a sixtuple $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$, 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/hidden labels respectively, containing the distinct quiescence label $\delta \in L_O$. We write $L = L_I \cup L_O \cup L_H$ for the set of all labels.
- $\Delta \subseteq S \times \mathit{Distr}(L \times S)$ is a finite transition relation such that for all input actions $a \in L_I$ and distributions $\mu \in \mathit{Distr}(L \times S)$: $\mu(a, s') > 0$ implies $\mu(b, s'') = 0$ for all $b \neq a$ and some $s', s'' \in S$.

Example 3 Figure 2 presents two example pIOTSs and an invalid one. As by common convention we use "?" to suffix input and "!" to suffix output actions. By default, we let τ be an internal action. The target distribution of a transition is represented by a densely dotted arc between the edges belonging to it.

In Fig. 2a there is a non-deterministic choice between two inputs $a?$ and $b?$ modelling the choice that a user has in this state. If $a?$ is chosen, the automaton moves to state s_1. In case, the user chooses input $b?$, there is a 50% chance that the automaton moves to state s_2 and a 50% chance it moves to s_3. Note that the latter distribution is an example of an input-reactive distribution according to clause 4 in Definition 2.

On the contrary, state t_0 of Fig. 2b illustrates *output-generative* distributions. Output actions are not under the control of a user or the environment. Hence, in t_0 the system itself makes two choices: (1) it chooses one of the two outgoing distributions non-deterministically and (2) it chooses an output or internal action and the target state according to the chosen distribution. Note that both distributions are examples of output-generative distributions according to clause 4 in Definition 2.

Lastly, the rightmost model is not a valid pIOTS according to Definition 2 for two reasons: (1) There are two distinct input actions in one distribution and (2) input and output actions may not share one distribution, as both would violate clause 4 of Definition 2.

Notation. We make use of the following notations and concepts:

- Elements of the set of input actions are suffixed by "?" and elements of the set of output actions are suffixed by "!". By convention, we let τ represent an element of the set of internal actions.
- $s \xrightarrow{\mu, a} s'$ if $(s, \mu) \in \Delta$ and $\mu(a, s') > 0$ for some $s' \in S$,
- An action a is called *enabled* in a state $s \in S$, if there is an outgoing transition containing the label a. We write $s \rightarrow a$ if there are $\mu \in \mathit{Distr}(L \times S)$ and $s' \in S$ such that $s \xrightarrow{\mu, a} s'$ ($s \not\rightarrow a$ if not). The set of all enabled actions in a state $s \in S$ is denoted *enabled*(s).
- We write $s \xrightarrow{\mu, a}_{\mathcal{A}} s'$, etc. to clarify that a transition belongs to a pIOTS $\mathcal{A}$ if ambiguities arise.
- We call a pIOTS $\mathcal{A}$ *input enabled*, if all input actions are enabled in all states, i.e. for all $a \in L_I$ we have $s \rightarrow a$ for all $s \in S$.

Quiescence. In testing, a verdict must also be given if the system-under-test is quiescent, i.e. if it does not produce any output at all. Hence, the requirements model must explicitly indicate when quiescence is allowed and when not. This is expressed by a special output label δ, as required in clause 3. For more details on the treatment of quiescence we refer to Sect. 6 and for further reading to [STS13, Tre08].

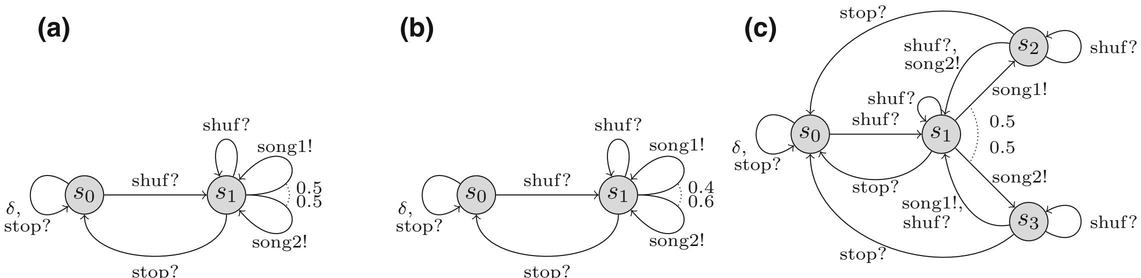


Fig. 3. Specification and two implementation pIOTSs of a shuffle music player. Some actions are separated by commas for readability indicating that two transitions with different labels are enabled from the same source to the same target states. **a** Specification, **b** unfair Implementation, **c** alternating Implementation

Example 4 Figure 3 shows three models of a simple shuffle mp3 player with two songs. The pIOTS in (3a) models the requirements: pressing the shuffle button enables the two songs with probability 0.5 each. The self-loop in s_1 indicates that after a song is chosen, both are enabled with probability 0.5 each again. Pressing the *stop* button returns the automaton to the initial state. Note that the system is required to be quiescent in the initial state until the shuffle button is pressed. This is denoted by the δ self-loop in state s_0.

The implementation pIOTS (3b) is subject to a small probabilistic deviation in the distribution over songs. Contrary to the requirements, this implementation chooses *song1* with a probability of 40% and gives a higher probability to *song2*.

In implementation (3c) the same song cannot be played twice in a row without intervention of the user or the environment. After the shuffle button is pressed, the implementation plays one song and moves to state s_2 or s_3 respectively. In these states only the respective other song is available.

Assuming that both incorrect models are hidden in a black box, the model-based testing framework presented in this paper is capable of detecting both flaws.

Parallel composition. The popularization of component based development demands an equivalent part on the modelling level. Individual components are designed and integrated later on. This notion is captured by the *parallel composition* of individual models.

Parallel composition is defined in the standard fashion [BKL08] by synchronizing on shared actions, and evolving independently on others. Since the transitions in the component pIOTSs are stochastically independent, we multiply the probabilities when taking shared actions, denoted by the operator $\mu \times \nu$. To avoid name clashes, we only compose *compatible* pIOTSs.

Note that parallel composition of two input-enabled pIOTSs yields a pIOTS.

Definition 5 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

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

- $S'' = S \times S'$,
- $L_I'' = \left(L_I \cup L_I'\right) \setminus \left(L_O \cup L_O'\right)$,
- $L_O'' = L_O \cup L_O'$,
- $L_H'' = L_H \cup L_H'$, and finally the transition relation
- $\Delta'' = \{((s,t), \mu) \in S'' \times Distr(L'' \times S'') \mid \mu \equiv \begin{cases} \nu_1 \times \nu_2 & \text{if } \exists\, a \in L \cap L' \text{ such that } s \xrightarrow{\nu_1, a} \wedge t \xrightarrow{\nu_2, a} \\ \nu_1 \times \mathbb{1} & \text{if } \forall\, a \in L \text{ with } s \xrightarrow{\nu_1, a} \text{ we have } t \not\rightarrow a \\ \mathbb{1} \times \nu_2 & \text{if } \forall\, a \in L' \text{ with } t \xrightarrow{\nu_2, a} \text{ we have } s \not\rightarrow a \end{cases}\}$,

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

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.

Paths. 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. We write $\pi' \sqsubseteq \pi$ to denote π' as a *prefix* of π, i.e. π' is finite and coincides with π on the first symbols of the sequence. The set of all finite paths of $\mathcal{A}$ is denoted by $\mathit{Paths}^{<\omega}(\mathcal{A})$ and all paths by $\mathit{Paths}(\mathcal{A})$.

Traces. 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 actions on its associated trace. All finite traces of $\mathcal{A}$ are summarized in $\mathit{Traces}^{<\omega}(\mathcal{A})$. The set of *complete traces*, $\mathit{cTraces}(\mathcal{A})$, contains every trace based on paths ending in deadlock states, i.e. states that do not enable any more actions. We write $\mathit{out}_{\mathcal{A}}(\sigma)$ for the set of output actions enabled in the states after trace σ.

2.3. Adversaries and trace distributions

Very much like traces 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 [Coh80] shows that it is impossible to assign a probability to all sets of traces, hence we use σ-fields consisting of cones. To illustrate the use of cones: 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.

Adversaries. Following the standard theory for probabilistic automata [Seg95], we define the behaviour of a pIOTS via adversaries (a.k.a. policies or schedulers) to resolve the non-deterministic choices; 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 6 An *adversary* E of a pIOTS $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$ is a function

$$E : \mathit{Paths}^{<\omega}(\mathcal{A}) \longrightarrow \mathit{Distr}(\mathit{Distr}(L \times S) \cup \{\perp\}),$$

such that for each finite path π, if $E(\pi)(\mu) > 0$, then $(\mathit{last}(\pi), \mu) \in \Delta$ or $\mu \equiv \perp$. We say that E is *deterministic*, if $E(\pi)$ assigns the Dirac distribution to every distribution for all $\pi \in \mathit{Paths}^{<\omega}(\mathcal{A})$. 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 $\mathit{Adv}(\mathcal{A}, k)$. The set of all adversaries of $\mathcal{A}$ is denoted $\mathit{Adv}(\mathcal{A})$.

Path probability. 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 $C_\pi = \{\pi' \in \mathit{Path}(\mathcal{A}) \mid \pi \sqsubseteq \pi'\}$. We use the inductively defined path probability function Q^E, i.e. $Q^E(s_0) = 1$ and

$$Q^E(\pi \mu a s) = Q^E(\pi) \cdot E(\pi)(\mu) \cdot \mu(a, s).$$

Note that an adversary E thus defines a unique probability measure P_E on the set of paths. Hence, the path probability function enables us to assign a unique probability space $(\Omega_E, \mathcal{F}_E, P_E)$ associated to an adversary E. Therefore, 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 7 The *trace distribution* D of an adversary $E \in Adv(\mathcal{A})$, denoted $D = trd(E)$ is the probability space $(\Omega_D, \mathcal{F}_D, P_D)$, where

1. $\Omega_D = L^\omega$,
2. $\mathcal{F}_D$ is the smallest σ-field containing the set $\{C_\beta \subseteq \Omega_D \mid \beta \in L^\omega\}$,
3. P_D is the unique probability measure on $\mathcal{F}_D$ such that $P_D(X) = P_E(trace^{-1}(X))$ for $X \in \mathcal{F}_D$.

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 $\mathcal{A} \sqsubseteq_{TD} \mathcal{B}$ 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 $(\Omega_D, \mathcal{F}_D, P_D)$ define probability spaces, follows from standard measure theory arguments, cf. [Coh80].

Example 8 Consider (c) in Fig. 3 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

$$\pi = s_0\ \mu_1\ \text{shuf?}\ s_1\ \mu_2\ \text{song1!}\ s_2\ \mu_3\ \text{shuf?}\ s_2 \quad \text{and} \quad \pi' = s_0\ \mu_1\ \text{shuf?}\ s_1\ \mu_2\ \text{song1!}\ s_2\ \mu_4\ \text{shuf?}\ s_1.$$

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 $P_{Trd(E)}(\sigma) = P_E(trace^{-1}(\sigma)) = P_E(\{\pi, \pi'\}) = \frac{1}{2}$.

3. Testing with probabilistic systems

Model-based testing entails the automatic test case generation, execution and evaluation based on a requirements model. We provide two algorithms for automated test case generation: an *offline* or *batch* algorithm, and an *online* or *on-the-fly* algorithm generating test cases during the execution. The first is used to generate batches of test cases before their execution, whereas the latter tests during the runtime of the system and evaluates on-the-fly.

Our goal is to test probabilistic systems based on a requirements specification. Therefore, the test procedure is split into two components; Functional testing and statistical hypothesis testing. The first assesses the functional correctness of the system under test, while the latter focuses on determining whether probabilities were implemented correctly.

The functional evaluation procedure is comparable to ones known from literature [NH84, TBS11]. Informally, we require all outputs produced by the implementation to be predictable by the requirements model. This condition is met by the input/output conformance (ioco) framework [Tre96], which we utilize in out theory.

Moreover, we present the evaluation procedure for the separate statistical verdict, assessing if probabilities were implemented correctly. Obviously, one test execution is not competent enough for that purpose and a large sample must be collected. Statistical methods and frequency analysis are then utilized on the gathered sample to give a verdict based on a chosen level of confidence.

3.1. Test generation and execution

Test cases. We formalize the notion of a (offline) test case over an action signature (L_I, L_O). Formally, a test case is a collection of traces that represent possible behaviour of a tester. These are summarized as a pIOTS in tree structure. The action signature describes the potential interaction of the test case with the SUT. In each state of a test, the tester can either provide some stimulus $a? \in L_I$, wait for a response $b! \in L_O$ of the system, or stop the overall testing process. When a test is waiting for a system response, it has to take into account all potential outputs including the situation that the system provides no response at all, modelled by δ, cf. Definition 2. [3]

[3] Note that in more recent version of *ioco* theory [Tre08], test cases are input-enabled. This enables them to catch possible outputs of the SUT before the test was able to supply the input. It can easily be incorporated into our framework.

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 [Gog00], but this is not supported in theory. Thus, our definition fills a small gap here.

Since the continuation of a test depends on the history, offline test cases are formalized as trees. For technical reasons, we swap the input and output label sets of a test case. This is to allow for synchronization/parallel composition in the context of input-reactive and output-generative transitions. We refer to Fig. 4 as an example.

Definition 9 A *test* or *test case* over an action signature (L_I, L_O) is a pIOTS of the form

$$t = \left(S^t, s_0^t, L_I^t, L_O^t, L_H^t, \Delta^t\right) := \left(S, s_0, L_O \setminus \{\delta\}, L_I \cup \{\delta\}, \left\{\tau_{obs}, \tau_{stim}, \tau_{stop}\right\}, \Delta\right)$$

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
  - $\mathit{enabled}(s) = \emptyset$, or
  - $\mathit{enabled}(s) = \left\{\tau_{obs}, \tau_{stim}, \tau_{stop}\right\}$, or
  - $\mathit{enabled}(s) = L_I^t \cup \{\delta\}$, or
  - $\mathit{enabled}(s) \subseteq L_O^t \setminus \{\delta\}$,

A *test suite* T is a set of test cases. A test case (suite resp.) for a pIOTS $\mathcal{S} = (S, s_0, L_I, L_O, L_H, \Delta)$, is a test case (suite resp.) over its action signature (L_I, L_O).

Test annotation. The next step is annotating the traces of a test with *pass* or *fail* verdicts determined by the requirements specification. Thus, annotating a trace pins down the behaviour, which we deem as acceptable or correct. This allows for automated evaluation of the functional behaviour. The classic *ioco* test case annotation suffices in that regard [TBS11]; Informally, a trace of a test case is labelled as *pass*, if it is present in the system specification and *fail* otherwise.

Definition 10 For a given test t a *test annotation* is a function

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

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 $\mathcal{S}$ with signature (L_I, L_O), we define the test annotation $a_{\mathcal{S},t} : \mathit{cTraces}(t) \longrightarrow \{\mathit{pass}, \mathit{fail}\}$ by

$$a_{\mathcal{S},t} = \begin{cases} \mathit{fail} & \text{if } \exists \varrho \in \mathit{Traces}^{<\omega}(\mathcal{S}), a \in L_O : \varrho\, a! \sqsubseteq \sigma \wedge \varrho\, a! \notin \mathit{Traces}^{<\omega}(\mathcal{S}) \\ \mathit{pass} & \text{otherwise.} \end{cases}$$

Example 11 Figure 4 shows two simple derived tests for the specification of a shuffle music player in Fig. 3. Note that the action signature is mirrored. This is to allow for synchronisation on shared actions according to Definition 5. Outputs of the test case are considered inputs for the SUT and vice versa. Since tests are pIOTSs, if $a!$ is an output action in the specification, there can only be $a?$-labelled input actions in one distribution in a test case due to the underlying input-reactive transitions.

The left side of Fig. 4 presents an annotated test case $\widehat{t}_1$, that is a classic test case according to the *ioco* test derivation algorithm [Tre96]. After the *shuffle* button is pressed, the test waits for a system response. Catching either *song1!* or *song2!* lets the test pass, while the absence of outputs yields the *fail* verdict.

The right side shows a probabilistic annotated test case $\widehat{t}_2$. We apply stimuli, observe, or stop with probabilities $\frac{1}{3}$ each. This is denoted by the probabilistic arc joining the three elements τ_{stim}, τ_{obs}, τ_{stop}. Moreover, the probabilistic choice over these three symbols illustrates how probabilities may help in steering the test process. After stimulating, we apply *stop!* and *shuf!* with probability $\frac{1}{2}$ each.

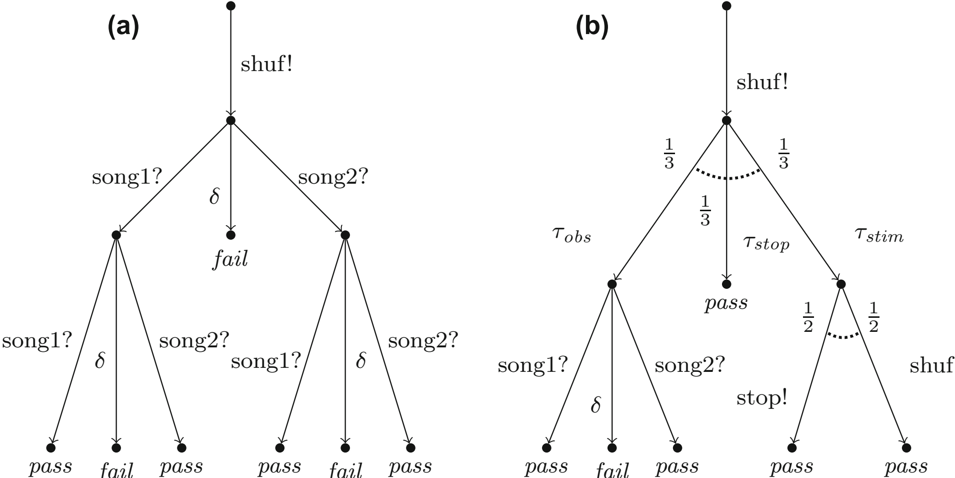


Fig. 4. Two annotated test cases derived from the specification of the shuffle mp3 player in Fig. 3. **a** Annotated test $\widehat{t_1}$, **b** annotated test $\widehat{t_2}$

Algorithm 1: Batch test generation for *pioco*.

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

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

```
   Input: Specification pIOTS 𝒮, an implementation 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 ℐ
5      σ := σ b!
6      if σ ∉ traces(𝒮) :
7          return fail
8      p_{σ,2} · [σ a? ∈ traces(𝒮)] →
9      try:
10         atomic
11             stimulate I with a?
12             σ := σ a?
13         end
14     catch output b! occurs before a? could be applied
15         σ := σ b!
16         if σ ∉ traces(𝒮) :
17             return fail
18     end
19 end
20 return pass
```

Algorithms. The recursive procedure *batch* in Algorithm 1 generates test cases, given a specification pIOTS $\mathcal{S}$ and a history σ, which is initially the empty history ϵ. Each step a probabilistic choice is made to return an empty test (line 2), to observe (line 4) or to stimulate (line 14), denoted with probabilities $p_{\sigma,1}$, $p_{\sigma,2}$ or $p_{\sigma,3}$ respectively. Note that we require $p_{1,\sigma} + p_{2,\sigma} + p_{3,\sigma} = 1$. This corresponds to clause 3 in Definition 9. A generated test case is concatenated with the result of *batch*. Thus, the procedure returns a pIOTS in tree shape. Recursively returning the empty test case in line 3 terminates a branch.

Lines 4–13 describe the step of observing the system; If a particular output is foreseen in the specification, it is added to the branch and the procedure *batch* is called again. If not, it is simply added to the branch. In the latter case, the branch of the tree stops and is to be labelled *fail*. Lines 14–23 refer to the stimulation of the system. An input action $a?$ present in the specification $\mathcal{S}$ is chosen. The algorithm adds additional branches, in case the system under test gives an output before stimulation takes place, i.e. lines 16–22.

Thus, Algorithm 1 continuously assembles a tree that terminates, depending on $p_{i,\sigma}$.

Algorithm 2 shows a sound way to generate and evaluate tests on-the-fly. It requires a specification $\mathcal{S}$, an implementation $\mathcal{I}$ and a test length $n \in \mathbb{N}$ as inputs. Initially, it starts with the empty history and concatenates an action label after each step. It terminates after n steps were executed (line 2).

Observing the system under test for outputs is reflected in lines 3–7. In case output or quiescence are observed, the algorithm checks whether this is allowed in the specification. If so, it proceeds with the next iteration and returns the *fail* verdict otherwise. Lines 8–18 describe the stimulation process. The algorithm tries to apply an input specified in the requirements. Should an output occur before this is possible, the algorithm evaluates the output like before.

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. Note that the choice of observing and stimulating depends on probabilities $p_{\sigma,1}$ and $p_{\sigma,2}$, where we require $p_{\sigma,1} + p_{\sigma,2} = 1$.

Theorem 12 *All test cases generated by Algorithm 1 are test cases according to Definition 9. All test cases generated by Algorithm 2 assign the correct verdict according to Definition 10.*

3.2. Test evaluation

In our framework, we assess functional correctness by the test verdict $a_{\mathcal{S},t}$ of Definition 10 and probabilistic correctness via further statistical analysis. While the first is straight forward, we elaborate on the latter in the following.

Statistical verdict. In order to reason about probabilistic correctness, a single test execution is insufficient. Rather, we collect a sample via multiple test runs. The sampling process consists of a push-button experiment in the sense of [Mil80]. Assume a black-box trace machine is given with input buttons, an action window and a reset button as illustrated in Fig. 5. An external observer records each individual execution before the reset button is pressed and the machine starts again. After a sample of sufficient size was collected, we compare the collected frequencies of traces to their expected frequencies according to the requirements specification. If the empiric observations are close to the expectations, we accept the probabilistic behaviour of the implementation.

Sampling. We set the parameters for sample length $k \in \mathbb{N}$, sample width $m \in \mathbb{N}$ and a level of significance $\alpha \in (0, 1)$. That is, we choose the length of individual runs, how many runs should be observed and a limit for the statistical *error of first kind*, i.e. the probability of rejecting a correct implementation.

Then, we check if the frequencies of the traces contained in this sample match the probabilities in the specification via statistical hypothesis testing. However, statistical methods can only be directly applied for purely probabilistic systems without non-determinism. Rather, we check if the observed trace frequencies can be explained, if we resolve non-determinism in the specification according to some scheduler. In other words, we hypothesize there is a scheduler that makes the occurrence of the sample likely.

Thus, during each run the black-box implementation $\mathcal{I}$ is governed by an unknown trace distribution $D \in \mathit{Trd}(\mathcal{I})$. In order for any statistical reasoning to work, we assume that D is the same in every run. Thus, the SUT chooses a trace distribution D and D chooses a trace σ to execute.

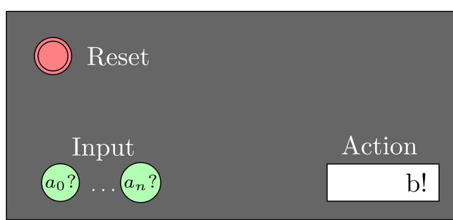
Frequencies and expectations. Our goal is to evaluate the deviation of a collected sample to the expected distribution. The function assessing the frequencies of traces within a sample $O = \{\sigma_1, \ldots, \sigma_m\}$ is given as a mapping $\mathit{freq} : \left(L^k\right)^m \to \mathit{Distr}\left(L^k\right)$, such that

$$\mathit{freq}\,(O)\,(\sigma) = \frac{|\{i=1,\ldots,m \wedge \sigma=\sigma_i\}|}{m}.$$

Hence, the function gives the relative frequency of a trace within a sample of size m.

To calculate the expected distribution according to a specification, we need to resolve all non-deterministic choices to get a purely probabilistic execution tree. Therefore, assume that a trace distribution D is given and k and m are fixed. We treat each run of the black-box as a Bernoulli trial. Recall that a Bernoulli trial has two outcomes: *success* with probability p and *failure* with probability $1 - p$. For each trace σ, we say that success occurred at position i if $\sigma = \sigma_i$, where σ_i is the i-th trace of the sample. Therefore, let $X_i \sim \mathit{Ber}\,(P_D\,(\sigma))$ be Bernoulli distributed random variables for $i = 1, \ldots, m$. Let $Z = \frac{1}{m}\sum_{i=1}^{m} X_i$ be the empiric mean with which we observe σ in a sample. Note that the expected probability under D then calculates as

$$\mathbb{E}^D\,(Z) = \mathbb{E}^D\left(\frac{1}{m}\sum_{i=1}^{m} X_i\right) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}^D\,(X_i) = P_D\,(\sigma)\,.$$



sampling →

ID	Recorded Trace σ	$\#\sigma$
σ_1	shuf? song1! song1!	15
σ_2	shuf? song1! song2!	24
σ_3	shuf? song2! song1!	26
σ_4	shuf? song2! song2!	35

Fig. 5. Black box trace machine with input alphabet $a_0?, \ldots, a_n?$, reset button and action window. Running the machine m times and observing traces of length k yields a sample. The ID together with the trace and the respective number of occurrences are noted down

Hence, the expected probability for each trace σ, is the probability that σ has, if the specification is governed by the trace distribution D.

Example 13 The right hand side of Fig. 5 shows a potential sample O that was collected from the shuffle music player of Fig. 3. The sample consists of $m = 100$ traces of length $k = 3$. In total there are 4 different traces with varying frequencies. For instance, the trace $\sigma_1 =$ *shuf? song1! song2!* has a frequency of $\mathit{freq}(O)(\sigma_1) = \frac{15}{100}$. Similarly, we calculate $\mathit{freq}(O)(\sigma_2) = \frac{24}{100}$, $\mathit{freq}(O)(\sigma_3) = \frac{26}{100}$ and $\mathit{freq}(O)(\sigma_4) = \frac{35}{100}$. Together, these frequencies form the empiric sample distribution.

Conversely, assume there is an adversary, that schedules *shuf?* with probability 1 and the distribution consisting of *song*1! and *song2*! with probability 1 in Fig. 3a. This adversary then induces a trace distribution D on the pIOTS of the shuffle-player. The expected probability of the observed traces under this trace distribution then calculates as $\mathbb{E}^D(\sigma_i) = 1 \cdot 1 \cdot 0.5 \cdot 0.5 = 0.25$ for $i = 1, \ldots, 4$.

The question we want to answer is, whether there exists a scheduler, such that the empiric sample distribution is sufficiently similar to the expected distribution.

Acceptable outcomes. The intuitive idea is to compare the sample frequency function to the expected distribution. If the observed frequencies do not deviate significantly from our expectations, we accept the sample. How much deviation is allowed depends on an a priori chosen level of significance $\alpha \in (0, 1)$.

We accept a sample O if $\mathit{freq}(O)$ lies within some distance r_α of the expected distribution $\mathbb{E}^D$. 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_α from the expected distribution are contained within the ball $B_{r_\alpha}(\mathbb{E}^D)$, where $\mathit{dist}(u, v) := \sup_{\sigma \in L^k} \mid u(\sigma) - v(\sigma) \mid$ and u and v are distributions. The set of all distributions together with the distance function thus define a metric space, and distance and deviation can be assessed. To limit the error of accepting an erroneous sample, we choose the smallest radius, such that the error of rejecting a correct sample is not greater than α by [4]

$$r_\alpha := \inf \left\{ r_\alpha \mid P_D\left(\mathit{freq}^{-1}\left(B_r(\mathbb{E}^D)\right)\right) > 1 - \alpha \right\}.$$

Definition 14 For $k, m \in \mathbb{N}$ and a pIOTS $\mathcal{A}$ the *acceptable outcomes* under $D \in \mathit{Trd}(\mathcal{A}, k)$ of significance level $\alpha \in (0, 1)$ are given by the set

$$\mathit{Obs}(D, \alpha, k, m) = \left\{ O \in \left(L^k\right)^m \mid \mathit{dist}\left(\mathit{freq}(O), \mathbb{E}^D\right) \leq r_\alpha \right\}.$$

The set of *observations* of $\mathcal{A}$ is given by $\mathit{Obs}(\mathcal{A}, \alpha, k, m) = \bigcup_{D \in \mathit{Trd}(\mathcal{A}, k)} \mathit{Obs}(D, \alpha, k, m)$.

The set of acceptable outcomes consists of all possible samples that we are willing to accept as close to our expectations, if the trace distributions D is given. Note that, due to non-determinism, the latter is required to make it possible to say what was expected in the first place. Since the choice of trace distributions depends on a scheduler that was chosen according to an unknown distribution, we sum up *all* acceptable outcomes as the set of observations.

The set of observations of a pIOTS $\mathcal{A}$ therefore has two properties, reflecting the error of false rejection and false acceptance respectively. If a sample was generated by a truthful trace distribution of the requirements specification, we correctly accept it with probability higher than $1 - \alpha$. Conversely, if a sample was generated by a trace distribution not admitted by the system requirements, the chance of erroneously accepting it is smaller than

[4] Note that $\mathit{freq}(O)$ is not a bijection, but used here for ease of notation.

some β_m. Here 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.

Goodness of fit. In order to state whether a given sample O is a truthful observation, we need to find a trace distribution $D \in Trd(\mathcal{A})$, such that $O \in Obs(D, m, k, \alpha)$. It guarantees that the error of rejecting a truthful sample is at most α. While the set of observations is crucial for the soundness and completeness proofs of our framework, they are computationally intractable to gauge for every D, since there are uncountably many.

To find the best fitting trace distribution in practice we resort to χ^2-hypothesis testing. The empirical χ^2 score is calculated as

$$\chi^2 = \sum_{i=1}^{m} \frac{(n(\sigma_i) - m \cdot \mathbb{E}^D(\sigma_i))^2}{m \cdot \mathbb{E}^D(\sigma_i)}, \tag{1}$$

where $n(\sigma)$ is the number with which σ occurred in the sample. The score can be understood as the cumulative sum of deviations from an expected value. Note that this entails a more general analysis of a sample than individual confidence intervals for each trace. The empirical χ^2 value is compared to critical values of given degrees of freedom and levels of significance. These values can be calculated or universally looked up in a χ^2 table.

Since expectations in our construction depend on a trace distribution to explain a possible sample, it is of interest to find the best fitting one. This turns (1) into an optimisation or constraint solving problem, i.e.

$$\min_D \sum_{i=1}^{m} \frac{\left(n(\sigma_i) - m \cdot \mathbb{E}^D(\sigma_i)\right)^2}{m \cdot \mathbb{E}^D(\sigma_i)}. \tag{2}$$

The probability of a trace is given by a scheduler and the corresponding path probability function, cf. Definition 6. Hence, by construction, we want to optimize the probabilities p used by a scheduler to resolve non-determinism. This turns (2) into a minimisation of a rational function $f(p)/g(p)$ with inequality constraints on the vector p. As shown in [NDG08], minimizing rational functions is **NP**-hard.

Optimization naturally finds the best fitting trace distribution. Hence, it gives an indication on the goodness of fit, i.e. how close to a critical value the empirical χ^2 value is. Alternatively, instead of finding the *best* fitting trace distribution one could turn (1) into a satisfaction or constraint solving problem in values of p. This answers if values of p exist such that the empirical χ^2 value lies below the critical threshold.

Example 15 Recall Example 13 and assume we want to find out, if the sample presented on the right in Fig. 5 is an observation of the specification of the shuffle music player, cf. Fig. 3a. We already established

$$freq(O)(\sigma_i) = \begin{cases} \frac{15}{100}, & \text{if } i = 1 \\ \frac{24}{100}, & \text{if } i = 2 \\ \frac{26}{100}, & \text{if } i = 3 \\ \frac{35}{100}, & \text{if } i = 4 \end{cases} \quad \text{and} \quad n(\sigma_i) = \begin{cases} 15, & \text{if } i = 1 \\ 24, & \text{if } i = 2 \\ 26, & \text{if } i = 3 \\ 35, & \text{if } i = 4. \end{cases}$$

If we fix a level of significance at $\alpha = 0.1$, the critical χ^2 value becomes $\chi^2_{crit} = 6.25$ for three degrees of freedom. Note that we have three degrees of freedom, since the probability of the fourth trace is implicitly given, if we know the rest.

Let E be an adversary, that schedules *shuf*? with probability p and the distribution consisting of *song1*! and *song2*! with probability q in Fig. 3a. We ignore the other choices the adversary has to make for the sake of this example. We are trying to find values for p and q such that the empiric χ^2 value is smaller than χ^2_{crit}, i.e.

$$\exists\, p, q \in [0, 1] : \frac{(15-100\cdot p\cdot q\cdot 0.25)^2}{100\cdot p\cdot q\cdot 0.25} + \frac{(24-100\cdot p\cdot q\cdot 0.25)^2}{100\cdot p\cdot q\cdot 0.25} + \frac{(26-100\cdot p\cdot q\cdot 0.25)^2}{100\cdot p\cdot q\cdot 0.25} + \frac{(35-100\cdot p\cdot q\cdot 0.25)^2}{100\cdot p\cdot q\cdot 0.25} < 6.25?$$

Using MATLABs [Gui98] function `fsolve()` for parameters p and q we quickly find the best empiric value as $\chi^2 = 8.08 > 6.25$. Hence, the minimal values for p and q provide a χ^2 minimum, which is still greater than the critical value. Therefore, there is no scheduler of the specification pIOTS that makes O a likely sample and we reject the potential implementation.

Contrary, assume Fig. 3b were the requirements specification, i.e. we require *song1!* to be chosen with only 40% and *song2!* with 60%. The satisfaction/optimisation problem for the same scheduler then becomes

$$\exists\, p, q \in [0,1] : \frac{(15-100\cdot p\cdot q\cdot 0.16)^2}{100\cdot p\cdot q\cdot 0.16} + \frac{(24-100\cdot p\cdot q\cdot 0.24)^2}{100\cdot p\cdot q\cdot 0.24} + \frac{(26-100\cdot p\cdot q\cdot 0.24)^2}{100\cdot p\cdot q\cdot 0.24} + \frac{(35-100\cdot p\cdot q\cdot 0.36)^2}{100\cdot p\cdot q\cdot 0.36} < 6.25,$$

because of different specified probabilities. In this scenario MATLABs [Gui98] `fsolve()` gives the best empiric χ^2 value as $\chi^2 = 0.257 < 6.25 = \chi^2_{crit}$ for $p = 1$ and $q = 1$. Hence, we found a scheduler that makes the sample O most likely and accept the potential implementation.

Verdict functions. With this framework, the following decision process summarizes if an implementation fails based on a functional and/or statistical verdict. An overall *pass* verdict is given to an implementation if and only if it passes both verdicts.

Definition 16 Given a specification $\mathcal{S}$, an annotated test $\hat{t}$ for $\mathcal{S}$, $k, m \in \mathbb{N}$ where k is given by the trace length of $\hat{t}$ and a level of significance $\alpha \in (0,1)$, we define the *functional verdict* as the function $v_{func} : pIOTS \longrightarrow \{pass, fail\}$, with

$$v_{func}(\mathcal{I}) = \begin{cases} pass & \text{if } \forall \sigma \in cTraces(\mathcal{I} \,||\, t) \cap cTraces(t) : a(\sigma) = pass \\ fail & \text{otherwise,} \end{cases}$$

the *statistical verdict* as the function $v_{stat} : pIOTS \longrightarrow \{pass, fail\}$, with

$$v_{stat}(\mathcal{I}) = \begin{cases} pass & \text{if } \exists D \in Trd(\mathcal{S}, k) : P_D\left(Obs\left(\mathcal{I} \,||\, \hat{t}, \alpha, k, m\right)\right) \geq 1-\alpha \\ fail & \text{otherwise,} \end{cases}$$

and finally the *overall verdict* as the function $V : pIOTS \rightarrow \{pass, fail\}$, with

$$V(\mathcal{I}) = \begin{cases} pass & \text{if } v_{func}(\mathcal{I}) = v_{stat}(\mathcal{I}) = pass \\ fail & \text{otherwise.} \end{cases}$$

An implementation passes a test suite $\hat{T}$, if it passes all tests $\hat{t} \in \hat{T}$.

The functional verdict is given based on the test case annotations, cf. Definition 10. The execution of a test case on the system under test is denoted by their parallel composition. Note that all given verdicts are correct, because the annotation is sound with respect to *ioco* [Tre08].

The statistical verdict is based on the sampling process. Therefore a test case has to be executed several times to gather a sufficiently large sample. A *pass* verdict is given, if the observation is likely enough under the best fitting trace distribution. If no such trace distribution exists, the observed behaviour cannot be explained by the requirements specification and the *fail* verdict is given.

Lastly, only if an implementation passes both the functional and statistical test verdicts, it is given the overall verdict *pass*.

4. Experimental validation

We show experimental results of our framework applied to three case studies known from the literature: (1) the Knuth and Yao Dice program [KY76], (2) the binary exponential backoff protocol [JDL02] and (3) the FireWire root contention protocol [SV99]. Our experimental set up can be seen in Fig. 6. We implemented these application using Java 7 and connected them to the MBT tool JTorX [Bel10]. JTorX was provided with a specification for each of the three case studies. It generated test cases of varying length for each of the applications and the results were saved in log files. For each application we run JTorX from the command line to initialize the random test generation algorithm with a new seed. In total we saved 10^5 log files for every application. None of the executed tests ended in a *fail* verdict for functional behaviour, i.e. all implementations appear to be functionally implemented correctly.

The statistical analysis was done using MATLAB [Gui98]. The function `fsolve()` was used for optimisation purposes in the parameters p, which represent the choices that the scheduler made. The statistical verdicts were calculated based on a level of significance $\alpha = 0.1$. Note that this gave the best fitting scheduler for each application to indicate the goodness of fit. We created mutants that implemented probabilistic deviations from the original protocols. All mutants were correctly given the statistical *fail* verdict and all supposed correct implementations yielded in statistical *pass* verdict.

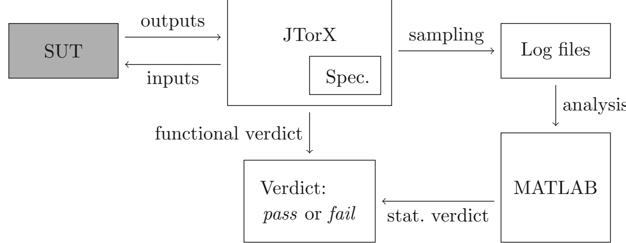


Fig. 6. Experimental set up entailing the system under test, the MBT tool JTorX [Bel10] and MATLAB [Gui98]

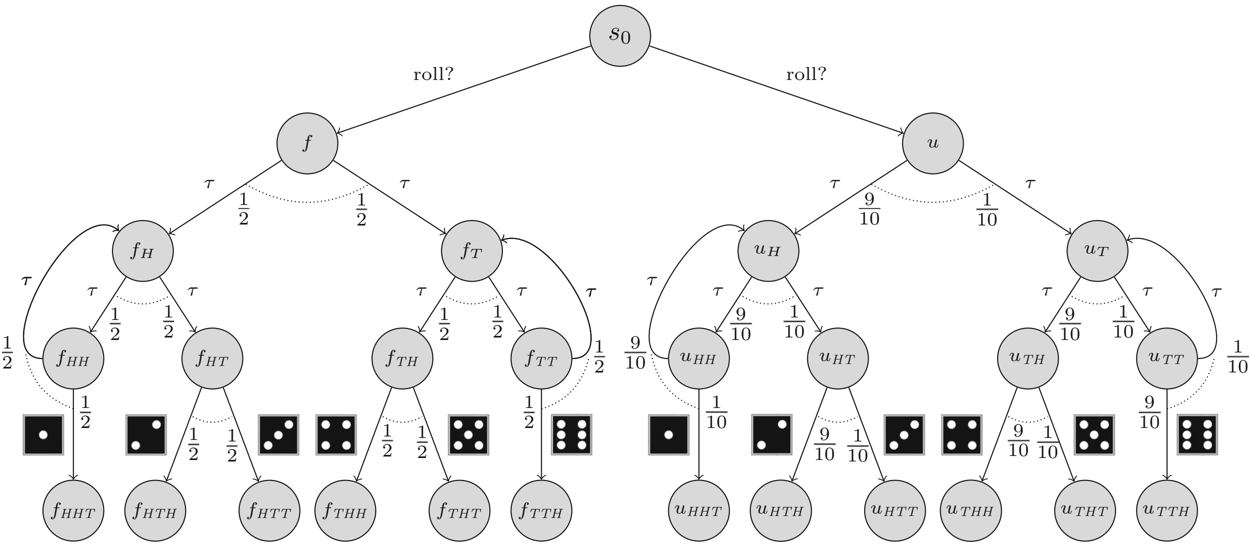


Fig. 7. Dice program based on Knuth and Yao [KY76]. The starting state enables a non-deterministic choice between a fair and an unfair die. The unfair die uses an unfair coin to determine its outcomes, i.e. the coin has a probability of 0.9 to yield *head*

4.1. Dice programs by Knuth and Yao

The dice programs by Knuth and Yao [KY76] aim at simulating a 6-sided die with multiple fair coin tosses. The uniform distribution on the numbers 1 to 6 is simulated by repeatedly evaluating the uniform distribution of the numbers 1 and 2 until an output is given. An example specification for a fair coin is given in Fig. 1.

Set up. To incorporate a non-deterministic choice we implemented a program that chooses between a fair die and an unfair (weighted) one. The unfair die uses an unfair coin to evaluate the outcome of the die roll. The probability to observe *head* with the unfair coin was set to 0.9. A model of the choice dice program can be seen in Fig. 7. The action *roll?* represents the non-deterministic choice of which die to roll. We implemented the application such that it chooses either die according to the current system time in milliseconds and added pseudo-random noise to avoid sampling over a simple probability distribution.

Results. We chose a level of significance $\alpha = 0.1$ and gathered a sample of 10^5 traces of length 2. We stored the logs for further statistical evaluation. The test process never ended due to erroneous functional behaviour. Consequently we assume that the implementation is functionally correct.

Table 1 presents the statistical results of our simulation and the expected probabilities if (1) the model KY1 of Fig. 1 is used as specification and (2) the model KY2 of Fig. 7 is used as specification. Since there is no non-determinism in KY1, we expect each value to have a probability of $\frac{1}{6}$.

Table 1. Observation of Knuth's and Yao's non-deterministic die implementation and their respective expected probabilities according to specification KY1 (cf. Fig. 1) or KY2 (cf. Fig. 7)

	⚀	⚁	⚂	⚃	⚄	⚅
Observed value	29473	29928	10692	12352	8702	8853
Relative frequency	0.294	0.299	0.106	0.123	0.087	0.088
Exp. probability KY1	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$
Exp. probability KY2	$\frac{p}{6} + \frac{(1-p)\cdot 81}{190}$	$\frac{p}{6} + \frac{(1-p)\cdot 81}{190}$	$\frac{p}{6} + \cdot \frac{(1-p)\cdot 9}{190}$	$\frac{p}{6} + \frac{(1-p)\cdot 81}{990}$	$\frac{p}{6} + \frac{(1-p)\cdot 9}{990}$	$\frac{p}{6} + \frac{(1-p)\cdot 9}{990}$

The parameter p depends on the scheduler that resolves the non-deterministic choice on which die to roll in KY2

In contrast, there is a non-determinisic choice to be resolved in KY2. Hence, the expected value is given depending on the parameter p, i.e. the probability with which the fair or unfair die are chosen respectively. Note that we left out the *roll?* action in every trace of Table 1 for readability.

In order to assess if the implementation is correct with respect to a level of significance $\alpha = 0.1$, we compare the χ^2 value for the given sample to the critical one given by $\chi^2_{0.1} = 9.24$. The critical value can universally be calculated or looked up in any χ^2 distribution table. We use the critical value for 5 degrees of freedom, because the outcome of the sixth trace is determined by the respective other five.

KY1 as specification. The calculated score approximately yields $\chi^2_{KY1} = 31120 \gg 9.24 = \chi^2_{0.1}$. The implementation is therefore rightfully rejected, because the observation did not match our expectations.

KY2 as specification. The best fitting parameter p with MATLABs `fsolve()` yields $p = 0.4981$, i.e. the implementation chose the fair die with a probability of 49.81%. Consequently, a quick calculation showed $\chi^2_{KY2} = 5.1443 < 9.24 = \chi^2_{0.1}$. Therefore, the implementation is assumed to be correct, because we found a scheduler, that chooses the fair and unfair die such that the observation is likely with respect to $\alpha = 0.4$.

Our results confirm our expectations: The implementation is rejected, if we require a fair die only, cf. Fig. 1. However, it is accepted if we require a choice between the fair and the unfair die, cf. Fig. 7.

4.2. Binary Exponential Backoff algorithm in the IEEE 802.3.

The Binary Exponential Backoff protocol is a data transmission protocol between N hosts, trying to send information via one bus [JDL02]. If two hosts try to send at the same time, their messages collide and they pick a waiting time before trying to send their information again. After i collisions, the hosts randomly choose a new waiting time of the set $\{0, \ldots 2^i - 1\}$ until no further collisions take place. Note that information thus gets delivered with probability one since the probability of infinitely many collisions is zero.

Set up. We implemented the protocol in Java 7 and gathered a sample of 10^5 traces of length 5 for two communicating hosts. Note that the protocol is only executed if a collision between the two hosts arises. Therefore, each trace we collect starts with the *collide!* action. This is due to the fact that the two hosts initially try to send at the same time, i.e. time unit 0. If a host successfully delivers its message it acknowledges this with the *send!* output and resets its clock to 0 before trying to send again.

Our specification of this protocol does not contain non-determinism. Thus, calculations in this example were not subject to optimization or constraint solving to find the best fitting scheduler/trace distribution.

Results. The gathered sample is displayed in Table 2. The values of n show how many times each trace occurred. For comparison, the value $m \cdot \mathbb{E}(\sigma)$ gives the expected number according to our specification of the protocol. Here, m is the total sample size and $\mathbb{E}(\sigma)$ the expected probability. The interval $[l_{0.1}, r_{0.1}]$ was included for illustration purposes and represents the 90% confidence interval under the assumption that the traces are normally distributed. It gives a rough estimate on how much values are allowed to deviate for the given level of confidence $\alpha = 0.1$.

Table 2. A sample of the binary exponential backoff protocol for two communicating hosts

ID	Trace σ	n	$\approx m\mathbb{E}(\sigma)$	$[l_{0.1}, u_{0.1}]$	$\approx \frac{(n-m\mathbb{E}(\sigma))^2}{m\mathbb{E}(\sigma)}$
1	collide! send! collide! send! send!	18,656	18,750	[18592, 18907]	0.47
2	collide! send! collide! send! collide!	18,608	18,750	[18592, 18907]	1.08
3	collide! collide! send! collide! send!	16,473	16,408	[16258, 16557]	0.26
4	collide! collide! send! send! collide!	12,665	12,500	[12366, 12633]	2.18
5	collide! send! collide! collide! send!	11,096	10,938	[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*

We collected a total of $m = 10^5$ traces of length $k = 5$. Calculations yield $\chi^2 = 14.84 < 17.28 = \chi^2_{crit} = \chi^2_{0.1}$, hence we accept the implementation

However, we are interested in the multinomial deviation, i.e. less deviation of one trace allows higher deviation for another trace. In order to assess the statistical correctness, we compare the critical value χ^2_{crit} to the empiric χ^2 score. The first is given as $\chi^2_{crit} = \chi^2_{0.1} = 17.28$ for $\alpha = 0.1$ and 11 degrees of freedom. This value can universally be calculated or looked up in a χ^2 distribution table. The empirical value is given by the sum of the entries of the last column of Table 2.

A quick calculation shows $\chi^2 = 14.84 < 17.28 = \chi^2_{0.1}$. Consequently, we have no statistical evidence that hints at wrongly implemented probabilities in the backoff protocol. In addition, the test process never ended due to a functional fail verdict. Therefore, we assume that the implementation is correct.

4.3. IEEE 1394 FireWire Root Contention Protocol

The IEEE 1394 FireWire Root Contention Protocol [SV99] elects a leader between two contesting 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 sends out a message itself, asking the other node to be the leader. Thus, the four possible 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 [SV99] as it is not clear, which node flips its coin first. Further, if different times were picked, e.g., $fast_1$ and $slow_2$, 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.

Set up. We implemented the root contention protocol in Java 7 and created four probabilistic mutants of it. The correct implementation C utilizes fair coins to determine the waiting time before it sends a message. The mutants M_1, M_2, M_3 and M_4 were subject to probabilistic deviations giving advantage to the second node via:

Mutant 1. $P(fast_1) = P(slow_2) = 0.1$,

Mutant 2. $P(fast_1) = P(slow_2) = 0.4$,

Mutant 3. $P(fast_1) = P(slow_2) = 0.45$ and

Mutant 4. $P(fast_1) = P(slow_2) = 0.49$.

Table 3. A sample of length $k = 5$ and depth $m = 10^5$ of the FireWire root contention protocol. Calculations of χ^2 are done after optimization in the parameter p. It represents which node got to flip its coin first

ID	Trace σ	$\approx m\mathbb{E}^D(\sigma)(p)$	n_c	n_{M_1}	n_{M_2}	n_{M_3}	n_{M_4}
1	c_1? slow$_1$! c_2? slow$_2$! retry!	$6250 \cdot p$	3148	1113	3091	3055	3161
2	c_1? slow$_1$! c_2? slow$_2$! done!	$18{,}750 \cdot p$	9393	3361	9047	9242	9329
3	c_1? slow$_1$! c_2? fast$_2$! done!	$25{,}000 \cdot p$	12,531	40,507	18,163	15,129	12,982
4	c_1? fast$_1$! c_2? fast$_2$! retry!	$8333 \cdot p$	4254	1467	4037	4066	4179
5	c_1? fast$_1$! c_2? fast$_2$! done!	$16{,}667 \cdot p$	8227	3048	7858	8474	8444
6	c_1? fast$_1$! c_2? slow$_2$! done!	$25{,}000 \cdot p$	12,438	504	7918	10,128	11,867
7	c_2? slow$_2$! c_1? slow$_1$! retry!	$6250 \cdot (1-p)$	3073	1137	2961	3256	3135
8	c_2? slow$_2$! c_1? slow$_1$! done!	$18{,}750 \cdot (1-p)$	9231	3427	9069	9456	9368
9	c_2? slow$_2$! c_1? fast$_1$! done!	$25{,}000 \cdot (1-p)$	12,657	447	8055	9685	11,975
10	c_2? fast$_2$! c_1? fast$_1$! retry!	$8333 \cdot (1-p)$	4211	1466	4008	4131	4199
11	c_2? fast$_2$! c_1? fast$_1$! done!	$16{,}667 \cdot (1-p)$	8335	2977	7969	8295	8312
12	c_2? fast$_2$! c_1? slow$_1$! done!	$25{,}000 \cdot (1-p)$	12,502	40,546	17,824	15,083	13,049
		$p_{opt} \approx$	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*

Statistically, the mutants should declare node 1 the leader more frequently. This is due to the fact that node 2 sends a leadership request faster on average.

Results. Table 3 shows the 10^5 recorded traces of length 5, where c_1? and c_2? denote the coins of node 1 and node 2 respectively. The expected value $\mathbb{E}^D(\sigma)$ depends on resolving one non-deterministic choice by varying p (which coin was flipped first). Note that the second non-deterministic choice was not subject to optimization, but immediately clear by the collected trace frequencies.

The empirical χ^2 score was calculated depending on parameter p and compared to the critical value χ^2_{crit}. The latter is given as $\chi^2_{crit} = \chi^2_{0.1} = 17.28$ for $\alpha = 0.1$ and 11 degrees of freedom. We used MATLABs `fsolve()` to find the optimal value for p, such that the empirical value χ^2 is minimal. The resulting verdicts can be found in the last row of Table 3. We can see that the only accepted implementation was C, because $\chi^2_C < 17.28$, whereas $\chi^2_{M_i} \gg 17.28$ for $i = 1, \ldots, 4$.

5. 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. However, soundness and completeness require a formal definition of what *correctness* entails. Hence, we formalize a *probabilistic input/output conformance* (pioco) relation. The *pioco*-relation pins down mathematically which system is allowed to subsume the other.

5.1. Probabilistic input/output conformance $\sqsubseteq_{pioco}$

The classical *ioco* relation [Tre96] states that an implementation conforms to the requirements, if it never provides any unspecified output or quiescence. Thus, the set of output actions after a trace of an implementation should be contained in the set of output actions after the same trace of the specification. However, instead of checking for all possible traces, the *ioco*-relation only checks traces that were specified in the requirements. As argued in [Tre96] this is important for implementation freedom.

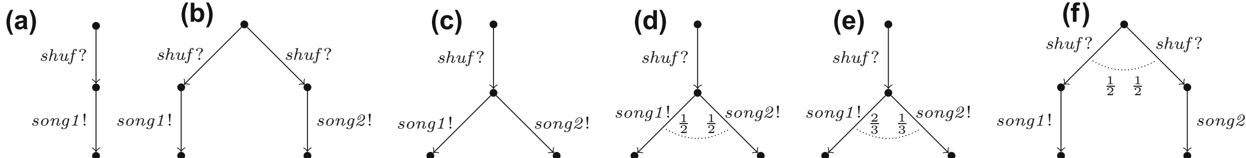


Fig. 8. Yardstick examples of a simplified shuffle player illustrating *pioco*. The three leftmost models represent regular IOTSs, while the three rightmost examples are pIOTSs. We refer to Example 19 for more details. **a** $\mathcal{A}_1$, **b** $\mathcal{A}_2$, **c** $\mathcal{A}_3$, **d** $\mathcal{A}_4$, **e** $\mathcal{A}_5$, **f** $\mathcal{A}_6$

Mathematically for two IOTSs $\mathcal{I}$ and $\mathcal{S}$, with $\mathcal{I}$ input-enabled, we say $\mathcal{I} \sqsubseteq_{ioco} \mathcal{S}$, if and only if

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

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

1. the prefix relation for trace distributions $D \sqsubseteq_k D'$ is the analogue of trace prefixes, i.e. $D \sqsubseteq_k D'$ iff $\forall \sigma \in L^k$: $P_D(\sigma) = P_{D'}(\sigma)$, and
2. the *output continuation* trace distributions; these are the probabilistic counterpart of the set $out_\sigma(\mathcal{A})$. For a pIOTSs $\mathcal{A}$ and a trace distribution D of length k, the output continuation of D 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

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

Intuitively, an implementation should conform to a specification, if the probability of every trace in $\mathcal{I}$ specified in $\mathcal{S}$, 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. Hence, the subset relation is violated and the implementation is not *pioco* conform to the requirements.

Definition 17 Let $\mathcal{I}$ and $\mathcal{S}$ be two pIOTSs. Furthermore let $\mathcal{I}$ be input-enabled, then we say $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$ iff

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

The *pioco* relation conservatively extends the *ioco* relation, i.e. both relations coincide for IOTSs. Recall that a pIOTS essentially is an input output transition system with transitions having distributions over states as target. Conversely, an IOTS can be treated as pIOTS where every distribution is the Dirac distribution, i.e. a distribution with a unique target.

Theorem 18 *Let $\mathcal{A}$ and $\mathcal{B}$ be two IOTSs and $\mathcal{A}$ be input-enabled, then*

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

Example 19 In Fig. 8 we present six toy examples to illustrate the *pioco* relation. Note that the three leftmost examples do not utilize a probabilistic transition other than the Dirac distribution over the target state. They can therefore be interpreted as regular IOTSs. The three rightmost models utilize probabilistic transitions, e.g., there is a probabilistic choice between *song1!* and *song2!* with probability 0.5 each in $\mathcal{A}_4$.

The original *ioco* relation checks whether the output of an implementation, after a specified trace, was expected. To illustrate, $\mathcal{A}_1$ is *ioco* conform to both $\mathcal{A}_2$ and $\mathcal{A}_3$. The input *shuf?* yields the output *song1!*, which is a subset of what was specified by the latter two, i.e. {*song1!*, *song2!*}. Note that it is irrelevant if the non-deterministic choice is over the *shuf?* actions or the output actions.

The *pioco* relation combines probabilistic and non-deterministic choices. $\mathcal{A}_4$ and $\mathcal{A}_5$ are not *pioco* for using different probabilities attached to the output actions. However, it is $\mathcal{A}_4 \sqsubseteq_{pioco} \mathcal{A}_3$. The requirements specification indicates a choice over *song1!* and *song2!*. If a system implements this choice with a 0.5 choice over the action, there is a scheduler in the specification that assigns exactly those probabilities to the actions *song1!* and *song2!*. Note that the opposite direction does not hold, because any implementation would need to assign probabilities 0.5 to each action, while the non-determinism indicates a free choice of probabilities.

For a complete list of the conformances in Fig. 8 we refer to Table 4.

Table 4. Illustration of the *pioco* relation with respect to Fig 8

pIOTS	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$	$\mathcal{A}_5$	$\mathcal{A}_6$
$\mathcal{A}_1$	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	–	–	–
$\mathcal{A}_2$	–	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	–	–	–
$\mathcal{A}_3$	–	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	–	–	–
$\mathcal{A}_4$	–	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	–	–
$\mathcal{A}_5$	–	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	–	$\sqsubseteq_{pioco}$	–
$\mathcal{A}_6$	–	$\sqsubseteq_{pioco}$	$\sqsubseteq_{pioco}$	–	–	$\sqsubseteq_{pioco}$

Note that *pioco* coincides with *ioco* for IOTSs, i.e. Figs. 8a–c

A tester must be able to apply every input at any given state of the SUT. This is reflected in classic *ioco*-theory by always assuming the implementation to be input enabled, cf. [Tre96]. If the specification is input-enabled too, then *ioco* coincides with trace inclusion. We show that *pioco* coincides with trace distribution inclusion in the pIOTS case. Moreover, our results show that *pioco* is transitive, just like *ioco*.

Theorem 20 *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 $\mathcal{A} \sqsubseteq_{pioco} \mathcal{C}$.

5.2. Soundness and completeness

Talking about soundness and completeness when referring to probabilistic systems is not a trivial topic, since one of the main difficulties of statistical analysis is the possibility of false rejection or false acceptance. This means that the application of null hypothesis testing inherently includes the possibilities to erroneously reject a true hypothesis or to falsely accept an invalid one by chance.

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. 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, as reflected in the verdicts we assign, cf. Definition 16.

Definition 21 Let $\mathcal{S}$ be a specification over an action signature (L_I, L_O), $\alpha \in (0, 1)$ be the level of significance and $\widehat{T}$ an annotated test suite for $\mathcal{S}$. Then

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

$$\mathcal{I} \sqsubseteq_{pioco} \mathcal{S} \Longrightarrow V(\mathcal{I}) = pass.$$

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

$$\mathcal{I} \not\sqsubseteq_{pioco} \mathcal{S} \Longrightarrow V(\mathcal{I}) = fail.$$

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 22 (Soundness) *Each annotated test for a pIOTS $\mathcal{S}$ is sound for every level of significance $\alpha \in (0, 1)$ with respect to 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, cf. Definition 14.

Theorem 23 (Completeness) *The set of all annotated test cases for a specification $\mathcal{S}$ is complete for every level of significance $\alpha \in (0, 1)$ with respect to pioco.*

6. Divergence and well-formed systems

Quiescence is a crucial concept in modelling system behaviour, since it explicitly represents the fact that no output is produced. If an implementation does not provide any output a test evaluation algorithm must decide whether or not this behaviour is acceptable. To illustrate, imagine a coffee machine does not provide coffee after money was inserted. On the contrary, it should not supply anything if no one interacts with it. In Definition 2 we explicitly include the special label δ to represent quiescence in a model.

Additionally, we allow internal actions in pIOTSs. This set of action labels is used in the literature to either model the invisible progress of system components or to hide actions which are not important for the current analysis. However, with invisible progress comes the possibility of divergent systems. A divergent system entails the occurrence of infinitely many internal actions, for instance represented by a self-loop labelled with τ in a state. Since this action is assumed to be internal and therefore invisible, an external observer might assume the system to be quiescent, even though it makes progress.

Stokkink et al. [STS13] were first in treating divergent and quiescent states as first class citizens of a model and introduced properties that a system must satisfy in order to be called *well-formed*. Under certain circumstances a well-formed system treats divergent states as quiescent, thus enabling a clean theoretical framework for model-based testing with quiescence.

We adapt their four rules to account for divergent systems in our probabilistic test theory. This allows to model systems more naturally, since convergence is not required.

Divergence. We define the language theoretic concepts needed to define divergence of a system. Let π be an infinite path of the form

$$\pi = s_0 \ \mu_0 \ a_0 \ s_1 \ \mu_1 \ a_1 \ s_2 \ \mu_2 \ a_2 \ldots,$$

then π is called *state finite*, if it only traverses finitely many states, i.e. if the set $\{s_i\}_{i \in \mathbb{N}} \subseteq S$ is finite. We call π *fair*, if for any action $a \in L_O \cup L_H$ with infinitely many s_j in π for which $a \in \mathit{enabled}(s_j)$ there are infinitely many a in π. This means if a (subcomponent of a) system infinitely often wants to execute some of its actions, it will indeed infinitely often execute them. Note that we call finite paths fair by default. Lastly, an infinite fair path is called *divergent*, if all its actions are internal.

Definition 24 Let $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$ be a pIOTS. A state $s \in S$ is called *quiescent*, if it does not enable output or internal actions. The set of all quiescent states of $\mathcal{A}$ is denoted $q(S)$.

A state $s \in S$ is called *divergent*, if there is a state finite and fair, divergent path on which s occurs infinitely often. The set of all divergent states of $\mathcal{A}$ is denoted $d(\mathcal{A})$.

Example 25 Recall Knuth's and Yao's dice program in Fig. 1. The state f_{HH} is a divergent state, because there exists a state finite, fair divergent path of the form $\pi = f_{HH} \ \mu_1 \ \tau \ f_H \ \mu_2 \ \tau \ f_{HH} \ldots$. States $f_{HHH}, f_{HHT}, \ldots, f_{TTT}$ are quiescent states, because they do not enable any more internal or output actions.

Divergent paths in a pIOTS may yield quiescent observations in states that are not necessarily quiescent. Hence, the δ-action might be observed in non-quiescent states. This is due to an external observer not being able to differ between functional quiescence or infinitely many internal (invisible) actions. Stokkink et al. [STS13] phrase four rules for *well-formed* systems to formalise the semantics of quiescence incorporating divergent behaviour. We adapt them for pIOTSs.

Definition 26 A pIOTS $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$ is called *well-formed*, if it satisfies the following four rules for all $s, s', s'' \in S$:

Rule R1 Quiescence should be observable. We require quiescent or divergent states to have an outgoing δ transition, i.e.

$$\text{If } s \in q(\mathcal{A}) \text{ or } s \in d(\mathcal{A}), \text{ then } \delta \in \mathit{enabled}(s).$$

Rule R2 Quiescence ends in a quiescent state. There is no notion of timing in pIOTS, hence a quiescent observation is not associated with a particular duration. The execution of a δ-transition represents the indefinite absence of outputs. Enabling outputs after a quiescence observation is thus undesired. That is

$$\text{If } s \xrightarrow{\mu, \delta} s', \text{ for some } \mu, \text{ then } s' \in q(\mathcal{A}).$$

Rule R3 No new behaviour after a quiescence observation. Since there is no notion of timing involved in pIOTS, every behaviour that is possible after an observation of quiescence must also be present beforehand. That is

$$\text{If } s \xrightarrow{\mu,\delta} s', \text{ for some } \mu, \text{ then } Trd(s') \subseteq Trd(s)\,.$$

Rule R4 Continued quiescence preserves behaviour. Since quiescence represents the absence of output for an indefinite amount of time and pIOTS have no notion of timing, there should be no difference in observing quiescence once or multiple times. That is

$$s \xrightarrow{\mu,\delta} s' \text{ and } s \xrightarrow{\nu,\delta} s'' \text{ for some } \mu, \nu \text{ implies } Trd(s') = Trd(s'')$$

Here, $Trd(s)$ is the set of trace distributions, if s were the starting state of the pIOTS.

These four rules ensure that divergent behaviour is correctly accounted for in pIOTS with respect to an external observer. Despite the fact that we impose additional rules for the design of well-formed pIOTSs, the model becomes more intuitive. The authors of [STS13] provide an algorithm that turns any IOTS into a well-formed one. The following theorem guarantees the existence of such a well-formed pIOTS using their approach.

Theorem 27 *Given a pIOTS $\mathcal{A}$ with $\delta \notin L_O$ such that all divergent paths are state-finite, there exists a unique, well-formed pIOTS $\delta(\mathcal{A})$ corresponding to $\mathcal{A}$.*

To allow for a component based design approach of models, we show that the composition of well-formed system is again well-formed. As such, a bottom-up design process that incorporates quiescent and divergent behaviour is possible in our framework.

Theorem 28 *The parallel composition of input enabled, compatible and well-formed pIOTSs is well-formed.*

7. Conclusions

We defined a sound and complete framework to test probabilistic systems. At the core of our work is the conformance relation in the *ioco* tradition baptised *pioco*. We presented how to automatically derive, execute and evaluate probabilistic test cases from a requirements model. The evaluation process handles functional and statistical behaviour. While the first can be assessed by means of *ioco* theory, we utilize frequency statistics in the latter. Our soundness and completeness results show that the correct verdict can be assigned up to arbitrary precision by means of a level of significance for a sufficiently large sample.

We illustrated the application of our framework by means of three case studies from the literature: Knuth's and Yao's dice application, the binary exponential backoff protocol and the FireWire leadership protocol. The test evaluation process found no functional misbehaviour, indicating that the implemented functions were correct. Additionally, all correct implementations were given the statistical *pass* verdict, while all mutants were discovered.

Future work focuses on the practical aspects of our theory by providing tool support and larger case studies.

Appendix

Below, we present the proofs of our theorems numbered according to the occurrence in the paper.

Proofs.

Theorem 12. *All test cases generated by Algorithm 1 are test cases according to Definition 9. All test cases generated by Algorithm 2 assign the correct verdict according to Definition 10.*

Proof We refer to [TBS11] the proof of this theorem. The proof carries over directly, since our definitions of tests coincide, up to the probabilistic choices instead of non-deterministic ones. Moreover, the algorithms use the *ioco* test derivation, which is sound with respect to *ioco*. Since test cases are annotated correctly with respect to *ioco* and *pioco* uses the same annotation function, Algorithm 2 assigns the correct functional verdict. □

Theorem 18. *Let $\mathcal{A}$ and $\mathcal{B}$ be two IOTSs and $\mathcal{A}$ be input enabled, then*

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

Proof Let us first assume that the set of internal actions of both systems is empty.

$\boxed{\Longleftarrow}$ Let $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$ and $\sigma \in traces(\mathcal{B})$. Our goal is to show that $out_{\mathcal{A}}(\sigma) \subseteq out_{\mathcal{B}}(\sigma)$.

Assume that there is $b! \in out_{\mathcal{A}}(\sigma)$. We want to show that $b! \in out_{\mathcal{B}}(\sigma)$. For this, let $|\sigma| = k \in \mathbb{N}$ and $H \in trd(\mathcal{B}, k)$ such that $P_H(\sigma) = 1$, which is possible because $\sigma \in traces(\mathcal{B})$ and both $\mathcal{A}$ and $\mathcal{B}$ are IOTSs (i.e. non-probabilistic). The same argument gives us $outcont_{\mathcal{A}}(H) \neq \emptyset$, because $\sigma \cdot b! \in traces(\mathcal{A})$.

Consequently there is at least one trace distribution $H' \in outcont_{\mathcal{A}}(H)$ such that $P_{H'}(\sigma \cdot b!) > 0$. Let $\pi \in trace^{-1}(\sigma) \cap Path(\mathcal{B})$. We know $H' \in outcont_{\mathcal{B}}(H)$, because $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$ by assumption and thus there must be at least one $E' \in adv(\mathcal{B}, k+1)$ such that $trd(E') = H'$ and $Q^{E'}(\pi \cdot Dirac \cdot b!s') > 0$ for some $s' \in S_{\mathcal{B}}$. Hence, $b! \in enabled(last(\pi))$ and therefore $b! \in out_{\mathcal{B}}(\sigma)$.

$\boxed{\Longrightarrow}$ Let $\mathcal{A} \sqsubseteq_{ioco} \mathcal{B}$, $k \in \mathbb{N}$ and $H^* \in trd(\mathcal{B}, k)$. Assume that $H \in outcont_{\mathcal{A}}(H^*)$, then we want to show that we have $H \in outcont_{\mathcal{B}}(H^*)$.

Therefore let $E \in Adv(\mathcal{A}, k+1)$ such that $Trd(E) = H$. If we can find $E' \in Adv(\mathcal{B}, k+1)$ such that $Trd(E) = Trd(E')$, we are done. We will do this constructively in three steps.

1. By construction of H^* we know that there must be $E' \in Adv(\mathcal{B}, k+1)$, such that for all $\sigma \in L^k$ we have $P_{Trd(E')}(\sigma) = P_{H^*}(\sigma) = P_{Trd(E)}(\sigma)$. Thus $H^* \sqsubseteq_k trd(E')$.
2. We did not specify the behaviour of E' for path of length $k+1$. Therefore we choose E' such that for all traces $\sigma \in L^k$ and $a? \in L_I$ we have $P_{Trd(E')}(\sigma\, a?) = 0 = P_{Trd(E)}(\sigma\, a?)$. Note that this is possible because both $\mathcal{A}$ and $\mathcal{B}$ are IOTSs, i.e. non-probabilistic.
3. The last property to show is that $Trd(E) = Trd(E')$. Therefore let us now set the behaviour of E' for traces ending in outputs. Let $\sigma \in traces(\mathcal{A})$, then assume $a! \in out_{\mathcal{A}}(\sigma)$ and because $\mathcal{A} \sqsubseteq_{ioco} \mathcal{B}$, we know that $a! \in out_{\mathcal{B}}(\sigma)$.
   Now let $p := P_{Trd(E)}(\sigma) = P_{Trd(E')}(\sigma)$ and $q := P_{Trd(E)}(\sigma\, a!)$. By equality of the trace distributions for traces up to length k we know that $q \leq p \leq 1$ and therefore there is $\alpha \in [0, 1]$ s.t. $q = p\alpha$. Let $Path(\mathcal{B}) \cap trace^{-1}(\sigma) = \{\pi_1, \ldots, \pi_n\}$ for $n \in \mathbb{N}$. Without loss of generality, we choose E' such that

   $$E'(\pi_i)(Dirac) = \begin{cases} \alpha & \text{if } i = 1 \\ 0 & \text{else} \end{cases}.$$

   In this way we have constructed $E' \in Adv(\mathcal{B}, k+1)$, such that for all $\sigma \in L^{k+1}$ we have $P_{Trd(E')}(\sigma) = P_{Trd(E)}(\sigma)$ and thus $Trd(E) = Trd(E')$, which finally yields $H \in outcont_{\mathcal{B}}(H^*)$.
   The above proof runs completely analogously if the set of internal actions of both systems is not empty. □

Lemma A. *Let $\mathcal{A}$ and $\mathcal{B}$ be two input-enabled pIOTSs, then*

$$\mathcal{A} \sqsubseteq_{TD} \mathcal{B} \Longrightarrow \mathcal{A} \sqsubseteq_{pioco} \mathcal{B}.$$

Proof Let $\mathcal{A} \sqsubseteq_{TD}^k \mathcal{B}$ then for every $H \in Trd(\mathcal{A}, k)$ we also have $H \in Trd(\mathcal{B}, k)$. Pick $m \in \mathbb{N}$, let $H^* \in Trd(\mathcal{B}, m)$ and take $H \in outcont_{\mathcal{A}}(H^*) \subseteq Trd(\mathcal{A}, m+1)$. We want to show that $H \in outcont_{\mathcal{B}}(H^*)$.

By assumption we know that $H \in trd(\mathcal{B}, m+1)$. In particular that means there must be at least one adversary $E \in adv(\mathcal{B}, m+1)$ such that $Trd(E) = H$. However, for this adversary, we know that $H^* \sqsubseteq_m Trd(E)$ and for all $\sigma \in L^m L_I$ we have $P_{Trd(E)}(\sigma) = 0$ and by trace distribution inclusion $Trd(E) = H$. Thus $H \in outcont_{\mathcal{B}}(H^*)$ and therefore $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$. □

Theorem 20. *Let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be pIOTSs and let $\mathcal{A}$ and $\mathcal{B}$ be input enabled, then*

(i) $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$ *if and only if* $\mathcal{A} \sqsubseteq_{TD} \mathcal{B}$.
(ii) $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$ *and* $\mathcal{B} \sqsubseteq_{pioco} \mathcal{C}$ *then* $\mathcal{A} \sqsubseteq_{pioco} \mathcal{C}$.

Proof $\boxed{(i)}$ $\boxed{\Longrightarrow}$ Assume $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$. Let $m \in \mathbb{N}$ and $D \in Trd(\mathcal{A}, m)$. We need to show that $D \in Trd(\mathcal{B}, m)$. The proof is by induction. Let $m = 0$, then obviously $D \in Trd(\mathcal{B}, 0)$. Now assume $Trd(\mathcal{A}, n) \subseteq Trd(\mathcal{B}, n)$ was shown for some n. We show that it holds for $m = n+1$. First, choose $D' \sqsubseteq_n D$ and note that $D' \in Trd(\mathcal{B}, n)$. Then, by assumption we know $outcont_{\mathcal{A}}(D') \subseteq outcont_{\mathcal{B}}(D')$. In particular, we know there exists $D'' \in Trd(\mathcal{B}, m)$, such that

$$\forall \sigma \in L^n \cdot L_O : P_D(\sigma) = P_{D''}(\sigma). \tag{3}$$

Since both $\mathcal{A}$ and $\mathcal{B}$ are input enabled, we also know there exists a trace distribution, say $D''' \in Trd(\mathcal{B}, m)$ for which (3) holds and additionally

$$\forall \sigma \in L^n \cdot L_I : P_D(\sigma) = P_{D''}(\sigma). \tag{4}$$

With both (3) and (4), we know that $D \in Trd(\mathcal{B}, m)$. This completes the induction and consequently $\mathcal{A} \sqsubseteq_{TD} \mathcal{B}$. $\boxed{\Longleftarrow}$ See Lemma A for the proof. In particular we do not even require input enabledness for $\mathcal{B}$ in this case. $\boxed{(ii)}$ Let $\mathcal{A} \sqsubseteq_{pioco} \mathcal{B}$ and $\mathcal{B} \sqsubseteq_{pioco} \mathcal{C}$ and $\mathcal{A}$ and $\mathcal{B}$ be input-enabled. As shown earlier we know that $\mathcal{A} \sqsubseteq_{TD} \mathcal{B}$. So let $k \in \mathbb{N}$ and $H^* \in Trd(\mathcal{A}, k)$. Consequently also $H^* \in Trd(\mathcal{B}, k)$ and thus the following embedding holds

$$outcont_{\mathcal{A}}(H^*) \subseteq outcont_{\mathcal{B}}(H^*) \subseteq outcont_{\mathcal{C}}(H^*).$$

Consequently $\mathcal{A} \sqsubseteq_{pioco} \mathcal{C}$. □

Lemma B. *Let $\mathcal{A}$ and $\mathcal{B}$ be two pIOTSs, $\alpha \in (0, 1)$ and $k \in \mathbb{N}$. Then*

$$Trd(\mathcal{A}, k) \subseteq Trd(\mathcal{B}, k) \Longleftrightarrow \forall m \in \mathbb{N} : Obs(\mathcal{A}, \alpha, k, m) \subseteq Obs(\mathcal{B}, \alpha, k, m).$$

Proof The proof can be found in [CSV07] and is not further discussed here. □

Theorem 22. *(Soundness) Each annotated test for an input enabled pIOTS $\mathcal{S}$ is sound for every level of significance $\alpha \in (0, 1)$ with respect to pioco.*

Proof Let $\mathcal{I}$ be a pIOTS and $\hat{t}$ be a test for $\mathcal{S}$. Further assume that $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$. Then we want to show $V(\mathcal{I}) = pass$. By Definition 16 we have $V(\mathcal{I}) = pass$ if and only if $v_{func}(\mathcal{I}) = v_{stat}(\mathcal{I}) = pass$.

1. In order for $v_{func}(\mathcal{I}) = pass$, we need to show that $a^{pioco}_{\mathcal{S},t}(\sigma) = pass$ for all $\sigma \in exec_{\hat{t}}(\mathcal{I}) := ctraces\left(\mathcal{I} \,||\, \hat{t}\right) \cap ctraces\left(\hat{t}\right)$. Let $\sigma \in exec_{\hat{t}}(\mathcal{I})$. Furthermore let $\sigma_1 \in traces(\mathcal{S})$ and $a! \in L_O$ such that $\sigma_1 a! \sqsubseteq \sigma$. If no such σ_1 and $a!$ exist, we get $a^{pioco}_{\mathcal{S},t}(\sigma) = pass$, because then σ is a trace consisting solely of inputs. Our goal is to show $\sigma_1 a! \in traces(\mathcal{S})$. Let $|\sigma_1| = l$ be the length of σ_1. Obviously $\sigma_1 \in traces(\mathcal{I}) \cap traces(\mathcal{S})$ and together with assumption $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$, we choose $H \in Trd(\mathcal{S}, l)$ for which $outcont_{\mathcal{I}}(H) \neq \emptyset$. Without loss of generality, we choose $D' \in outcont_{\mathcal{I}}(H)$ such that $P_{D'}(\sigma_1 a!) > 0$. Finally, $D' \in outcont_{\mathcal{S}}(H)$, and thus $\sigma_1 a! \in traces(\mathcal{S})$. This gives $a^{pioco}_{\mathcal{S},t}(\sigma) = pass$ and ultimately $v_{func}(\mathcal{I}) = pass$.
2. In order for $v_{stat}(\mathcal{I}) = pass$ we need to show that

   $$\exists D \in Trd(\mathcal{S}, k) : P_D\left(Obs\left(\mathcal{I} \,||\, \hat{t}, \alpha, k, m\right)\right) \geq 1 - \alpha.$$

   By assumption $\mathcal{I} \sqsubseteq_{pioco} \mathcal{S}$ and with Theorem 20 also $\mathcal{I} \sqsubseteq_{TD} \mathcal{S}$. Since parallel composing two pIOTSs does not add transitions or change existing probabilities, we conclude $Trd\left(\mathcal{I} \,||\, \hat{t}\right) \subseteq Trd(\mathcal{I})$. Consequently $Trd\left(\mathcal{I} \,||\, \hat{t}\right) \subseteq Trd(\mathcal{S})$. Lemma B now shows for all $m \in \mathbb{N}$, that

   $$Obs\left(\mathcal{I} \,||\, \hat{t}, \alpha, k, m\right) \subseteq Obs(\mathcal{S}, \alpha, k, m). \tag{5}$$

By construction of the set of observations, cf. Definition 14 there is a trace distribution $D \in Trd\left(\mathcal{I} \,||\, \widehat{t}\right)$, such that

$$P_D\left(Obs\left(\mathcal{I} \,||\, \widehat{t}, \alpha, k, m\right)\right) \geq 1 - \alpha.$$

We utilize (5) to see

$$P_D\left(Obs\left(\mathcal{S}, \alpha, k, m\right)\right) \geq 1 - \alpha.$$

Finally, this yields $v_{stat}(\mathcal{I}) = pass$.

The functional part and the statistical part give $V(\mathcal{I}) = pass$. This means that an annotated test for $\mathcal{S}$ is sound with respect to $\sqsubseteq_{pioco}$. □

Theorem 23. *(Completeness) The set of all annotated test cases for a specification $\mathcal{S}$ is complete for every level of significance $\alpha \in (0,1)$ wrt pioco.*

Proof In order to show the completeness of test suite $\widehat{T}$ consisting of *all* tests, assume that $\mathcal{I} \not\sqsubseteq_{pioco} \mathcal{S}$. Our goal is to show that $v_{func}(\mathcal{I}) = fail$ or $v_{stat}(\mathcal{S}) = fail$. Since $\mathcal{I} \not\sqsubseteq_{pioco} \mathcal{S}$, there exists $k \in \mathbb{N}$ such that there is $D \in Trd(\mathcal{S}, k)$ for which

$$outcont_{\mathcal{I}}(D) \not\subseteq outcont_{\mathcal{S}}(D).$$

There are two cases to consider: (1) there is at least one trace of length $k+1$ ending in output, which is not in $traces(\mathcal{S})$ or (2) all traces of length $k+1$ ending in output are in $traces(\mathcal{S})$. These cases refer to the functional and probabilistic verdicts respectively. □

1. Assume there is at least one trace σ of length $k+1$ ending in output that is in $traces(\mathcal{I})$ but not in $traces(\mathcal{S})$. We need to show $v_{func}(\mathcal{I}) = fail$, or to be more specific, that $a_{\mathcal{S},t}(\sigma) = fail$. Since $\sigma \in L^k L_O$ and $\sigma \notin traces(\mathcal{S})$, we can conclude that there are $\sigma_1 \in traces(\mathcal{S})$ and $a! \in L_O$ such that $\sigma_1 a! \sqsubseteq \sigma$. Consequently, there is an annotated test $\hat{t} \in \widehat{T}$ with $\sigma \in traces(t)$. With $\sigma \in traces(\mathcal{I}) \cap ctraces(t)$ we conclude $\sigma \in exec_{\hat{t}}(\mathcal{I})$. Moreover $\sigma_1 \in traces(\mathcal{S})$ and $\sigma_1 a! \sqsubseteq \sigma$. Consequently $a_{\mathcal{S},t}(\sigma) = fail$. This gives $v_{func}(\mathcal{I}) = fail$.
2. Assume all traces of length $k+1$ ending in output are also in $traces(\mathcal{S})$. Then obviously $v_{func}(\mathcal{I}) = pass$. We need to show $v_{stat}(\mathcal{I}) = fail$, which requires

   $$\forall D \in Trd(\mathcal{S}, k+1) : P_D\left(Obs\left(\mathcal{I} \,||\, \widehat{t}, \alpha, k, m\right)\right) < 1 - \alpha$$

   for some $\widehat{t} \in \widehat{T}$ and $k, m \in \mathbb{N}$. By assumption, we know that

   $$\exists\, D \in Trd(\mathcal{I}, k+1)\, \forall\, D' \in Trd(\mathcal{S}, k+1)\, \exists\, \sigma \in traces(\mathcal{I}) \cap traces(\mathcal{S}) \cap \left(L^k \cdot L_O\right) : P_D(\sigma) \neq P_{D'}(\sigma).$$

   We conclude $Trd(\mathcal{I}, k+1) \not\subseteq Trd(\mathcal{S}, k+1)$. Moreover, the set of trace distributions is closed [CSV07], therefore there is $\varepsilon > 0$, such that:

   $$\forall\, D' \in Trd(\mathcal{S}, k+1) : \min_{D \in Trd(\mathcal{I}, k+1)} \|\, D - D' \,\| \geq \varepsilon. \tag{6}$$

   By assumption, the probability of at least one σ cannot be matched. Hence, there must be a test $\widehat{t} \in \widehat{T}$ containing σ. Since $Trd\left(\mathcal{I} \,||\, \widehat{t}\right) \subseteq Trd(\mathcal{I})$ we estimate (6) further:

   $$\min_{D \in Trd\left(\mathcal{I} || \widehat{t}, k+1\right)} \|\, D - D' \,\| \geq \min_{D \in Trd(\mathcal{I}, k+1)} \|\, D - D' \,\| \geq \varepsilon.$$

   By construction of the set of observations, cf. Definition 14, we know that for all $D \in Trd\left(\mathcal{I} \,||\, \widehat{t}, k\right)$ we have $P_D\left(Obs\left(D', \alpha, k, m\right)\right) \leq \beta(\alpha, \varepsilon, m)$, where $D' \in Trd(\mathcal{S}, k)$. Now iff $\alpha < 1 - \beta(\alpha, \varepsilon, m)$, we get for all $k, m \in \mathbb{N}$:

   $$\max_{D \in Trd(\mathcal{S}, k+1)} P_D\left(Obs\left(\mathcal{I} \,||\, \widehat{t}, \alpha, k, m\right)\right) \leq \beta(\alpha, \varepsilon, m) < 1 - \alpha.$$

   However, with $\alpha \in (0,1))$ and $\beta(\alpha, \varepsilon, m) \xrightarrow{m \to \infty} 0$ (note that ε and α are fixed), this will always hold for m sufficiently large. Thus, for $m' > m$, we have $v_{stat}(\mathcal{I}) = fail$. □

Theorem 27. *Given a pIOTS $\mathcal{A}$ with $\delta \notin L_O$ such that all divergent paths are state-finite, there exists a unique, well-formed pIOTS $\delta(\mathcal{A})$ corresponding to $\mathcal{A}$.*

Proof Let $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$ be a pIOTS with $\delta \notin L_O$ such that all divergent paths in $\mathcal{A}$ are state finite. We define the system $\delta(\mathcal{A}) = (S_\delta, s_0, L_I, L_O \cup \{\delta\}, L_H, \Delta_\delta)$ as the system where

$$S_\delta = S \cup \left\{qos_s \text{ a state} \mid s \in d(\mathcal{A})\right\}$$

with qos_s being a newly introduced quiescence observation state of divergent states and

$$\begin{aligned}\Delta_\delta = \Delta & \cup \{(s, \mu) \in S \times Distr(\{\delta\} \times S) \mid \mu(\delta, s) = 1 \wedge s \in q(\mathcal{A})\} \\ & \cup \left\{(s, \mu) \in S \times Distr(\{\delta\} \times S_\delta) \mid \mu(\delta, qos_s) = 1 \wedge s \in d(\mathcal{A})\right\} \\ & \cup \left\{(qos_s, \mu) \in S_\delta \times Distr(\{\delta\} \times S_\delta) \mid \mu(\delta, qos_s) = 1 \wedge s \in d(\mathcal{A})\right\} \\ & \cup \left\{(qos_s, \mu) \in S \in Distr(L_I \times S) \mid s \in d(\mathcal{A}) \wedge s \xrightarrow{\mu, a?} s'\right\}.\end{aligned}$$

We show that $\delta(\mathcal{A})$ is a well-formed $\mathcal{A}$ according to Definition 26. In the following, let $Trd(s)_\delta$ be the set of trace distributions of $\delta(\mathcal{A})$ had the starting state $s \in S_\delta$. □

1. To prove that $\delta(\mathcal{A})$ satisfies Rule **R1**, we must show that for all states $s \in S_\delta$:

   If $s \in q(\delta(\mathcal{A}))$ or $s \in d(\delta(\mathcal{A}))$, then $\delta \in enabled(s)$.

   Since $s \in S_\delta \cap q(\mathcal{A})$ or $s \in S_\delta \cap d(\mathcal{A})$ following from the definition of $\delta(\mathcal{A})$ one of the three cases is possible: (a) $s \in S \cap q(\mathcal{A})$ (b) $s \in S \cap d(\mathcal{A})$ or (c) $s \in S_\delta \backslash S$ and $s \in q(\mathcal{A})$. Note that $s \in S_\delta \backslash S$ and $s \in d(\mathcal{A})$ is not possible by construction in $\delta(\mathcal{A})$.

   (a) Assume $s \in S \cap q(\delta(\mathcal{A}_\delta))$ holds. By definition of $\delta(\mathcal{A})$ no existing output or internal actions were removed or hidden. Hence, $s \in q(\mathcal{A})$. By definition of $\delta(\mathcal{A})$ we then have $(s, \mu) \in \Delta_\delta$ with $\mu(\delta, s) = 1$. Therefore $\delta \in enabled(s)$.
   (b) Assume $s \in S \cap d(\mathcal{A})$. That is to say s occurs infinitely often on a divergent path π in $\delta(\mathcal{A})$. Since $\delta(\mathcal{A})$ did not remove any actions or introduced additional internal actions, the divergent path π must also be present in $\mathcal{A}$. Therefore $s \in d(\mathcal{A})$. By definition of $\delta(\mathcal{A})$ we must therefore have $(s, \mu) \in \Delta_\delta$ such that $\mu(\delta, qos_s) = 1$, where qos_s is a newly introduced quiescence observation state in the set S_δ. Hence, $\delta \in enabled(s)$.
   (c) Assume $s \in S_\delta \backslash S$ and $s \in q(\delta(\mathcal{A}))$. That means s is a newly introduced quiescence observation state. By construction of $\delta(\mathcal{A})$ we have $\delta \in enabled(s)$.

2. To prove that $\delta(\mathcal{A})$ satisfies Rule **R2**, we must show that for all states $s, s' \in S_\delta$:

   If $s \xrightarrow{\mu, \delta} s'$ for some μ, then $s' \in q(\delta(\mathcal{A}))$.

   Again, there are three cases: (a) $s, s' \in S$ (b) $s \in S$ and $s' \in S_\delta \backslash S$ or (c) $s, s' \in S_\delta \backslash S$. Note that $s \in S_\delta \backslash S$ and $s' \in S$ is not possible by construction in $\delta(\mathcal{A})$.

   (a) Assume $s, s' \in S$ and $s \xrightarrow{\mu, \delta}_{\delta(\mathcal{A})} s'$. By definition of $\delta(\mathcal{A})$ we have $s = s'$. Hence s' is a quiescent state in $\delta(\mathcal{A})$.
   (b) Assume $s \in S$ and $s' \in S_\delta \backslash S$. By definition of $\delta(\mathcal{A})$ we know that s' is a quiescent observation state and therefore quiescent by construction.
   (c) Assume $s, s' \in S_\delta \backslash S$. From definition of $\delta(\mathcal{A})$ it follows that s' is a quiescent observation state and $s = s'$. Therefore s' is quiescent.

3. To prove that $\delta(\mathcal{A})$ satisfies Rule **R3** we must show that for all states $s, s' \in S_\delta$:

   If $s \xrightarrow{\mu, \delta} s'$ for some μ, then $Trd(s') \subseteq Trd(s)$.

   Since $s, s' \in S_\delta$ and $s \xrightarrow{\mu, \delta} s'$ for some μ it follows from definition $\delta(\mathcal{A})$ that the three possible cases are possible: (a) $s, s' \in S$ (b) $s \in S$ and $s' \in S_\delta \backslash S$ or (c) $s, s' \in S_\delta \backslash S$. Note that it is not possible that $s \in S_\delta \backslash S$ and $s' \in S$ and $s \xrightarrow{\mu, \delta} s'$ for some μ.

(a) Assume $s, s' \in S$. By definition of $\delta(\mathcal{A})$ we know that $s = s'$. Therefore $Trd(s') \subseteq Trd(s)$.

(b) Assume $s \in S$ and $s' \in S_\delta \backslash S$ with $s \xrightarrow{\mu,\delta} s'$ for some μ. By construction of $\delta(\mathcal{A})$ we know that $s \in d(\mathcal{A})$ and $s' \in q(\delta(\mathcal{A}))$, where the latter is a newly added quiescent state. Now assume $D \in Trd(s')$. Then we have two cases to consider: $D \in Trd(s', 0)$ or $D \in Trd(s', k)$ with $k \geq 1$. In the first case, we trivially have $D \in Trd(s)$ by definition. In the latter case, consider that all traces σ originating from s' are of the form $\sigma = a \cdot \sigma'$, where $a = \delta$ or $a \in L_I \cap enabled(s)$. Hence, by construction of $\delta(\mathcal{A})$ there exists an adversary in s whose trace distribution yields the same probabilities as the underlying adversary of D. Therefore $D \in Trd(s)$ and we find $Trd(s') \subseteq Trd(s)$.

(c) Assume that $s, s \in S_\delta$. By construction of $\delta(\mathcal{A})$, we know that s and s' are quiescent observation states and $s = s'$. Hence trivially $Trd(s') \subseteq Trd(s)$.

4. To prove that $\delta(\mathcal{A})$ satisfies Rule **R4**, we must show that for all states $s, s', s'' \in S_\delta$:

$$\text{If } s \xrightarrow{\mu,\delta} s' \text{ and } s' \xrightarrow{\nu,\delta} s'' \text{ for some } \mu, \nu, \text{ then } Trd\left(s'\right) = Trd\left(s''\right).$$

Since $s, s', s'' \in S_\delta$ and $s \xrightarrow{\mu,\delta} s'$ as well as $s' \xrightarrow{\nu,\delta} s''$ for some μ, ν, it follows from the definition of $\delta(\mathcal{A})$ that three cases are possible: (a) $s, s', s'' \in S$ (b) $s \in S$ and $s', s'' \in S_\delta \backslash S$ and (c) $s, s', s'' \in S_\delta \backslash S$. Note other cases are not possible by construction.

(a) Assume $s, s', s'' \in S$ and $s \xrightarrow{\mu,\delta} s'$ as well as $s' \xrightarrow{\nu,\delta} s''$ for some μ, ν. Then, by definition of $\delta(\mathcal{A})$ we know that $s = s' = s''$. Therefore obviously $Trd(s') = Trd(s'')$.

(b) Assume $s \in S$ and $s', s'' \in S_\delta \backslash S$ with $s \xrightarrow{\mu,\delta} s'$ as well as $s' \xrightarrow{\nu,\delta} s''$ for some μ, ν. Then by definition of $\delta(\mathcal{A})$ we know that s' is a quiescent observation state. By construction we know that $s' = s''$. Hence, clearly $Trd(s') = Trd(s'')$.

(c) Assume $s, s', s'' \in S_\delta \backslash S$. From definition of $\delta(\mathcal{A})$ we know that s is a quiescent observation state. Therefore $s = s' = s''$ and clearly $Trd(s') = Trd(s'')$.

This shows that $\delta(\mathcal{A})$ is a well-formed system for $\mathcal{A}$. □

Theorem 28. *The parallel composition of input enabled, compatible and well-formed pIOTSs is well-formed.*

Proof According to Definition 5 we have to prove that $\mathcal{A} \,||\, \mathcal{B}$ is well-formed, given two input enabled, well-formed and compatible pIOTSs of the form $\mathcal{A} = (S, s_0, L_I, L_O, L_H, \Delta)$ and $\mathcal{B} = \left(S', s'_0, L'_I, L'_O, L'_H, \Delta'\right)$. That is to say, we need to prove rules **R1** to **R4** hold for $\mathcal{A} \,||\, \mathcal{B}$.

1. In order to prove $\mathcal{A} \,||\, \mathcal{B}$ satisfies Rule **R1**, we must show for every state $(s, t) \in S''$:

$$\text{If } (s, t) \in q(\mathcal{A} \,||\, \mathcal{B}) \text{ or } (s, t) \in d(\mathcal{A} \,||\, \mathcal{B}), \text{ then } \delta \in enabled((s, t)).$$

Let $(s, t) \in S''$. We look at the cases of $(s, t) \in q(\mathcal{A} \,||\, \mathcal{B})$ and $(s, t) \in d(\mathcal{A} \,||\, \mathcal{B})$ separately. First assume $(s, t) \in q(\mathcal{A} \,||\, \mathcal{B})$. This means there is no $a \in L''_O \cup L''_H$ such that $a \in enabled((s, t))$. Since $\mathcal{A}$ and $\mathcal{B}$ are both input enabled, it follows from Definition 5 that there is no output or internal action enabled in s in the system $\mathcal{A}$ and no output or internal action enabled in t in the system $\mathcal{B}$. Hence, $s \in q(\mathcal{A})$ and $t \in q(\mathcal{B})$. Because both $\mathcal{A}$ and $\mathcal{B}$ are well-formed, we know that $\delta \in enabled(s)$ and $\delta \in enabled(t)$. By Definition 5 we conclude that $\delta \in enabled((s, t))$ in $\mathcal{A} \,||\, \mathcal{B}$. Next assume $(s, t) \in d(\mathcal{A} \,||\, \mathcal{B})$. That means there is an infinite, state-finite, fair divergent path π on which (s, t) occurs infinitely often. By Definition 5 each step of path π is a transition either by $\mathcal{A}$ or $\mathcal{B}$, since synchronisation on internal actions is not allowed. We conclude there are three cases to consider:

(a) Both $\mathcal{A}$ and $\mathcal{B}$ carry out an infinite number of internal transitions on path π

(b) $\mathcal{A}$ carries out a finite number of internal transitions and $\mathcal{B}$ carries out an infinite amount of internal transitions or

(c) $\mathcal{A}$ carries out a finite number of internal transitions and $\mathcal{A}$ carries out an infinite amount of internal transitions.

For each case, we will show that $\delta \in enabled(s)$ and $\delta \in enabled(t)$ from which $\delta \in enabled((s, t))$ follows. □

- Assume both $\mathcal{A}$ and $\mathcal{B}$ carry out an infinite amount of internal actions on path π. Without loss of generality assume that $\mathcal{A}$ is responsible for all even transitions, and $\mathcal{B}$ is responsible for all odd transitions, i.e.

  $$\pi = (s_1, t_1)\ \mu_1\ \ b_1\ \ (s_1, t_2)\ \mu_2\ \ a_1\ \ (s_2, t_2)\ \mu_3\ \ b_2\ \ (s_2, t_3)\ldots,$$

  where $s_i \in S$, $t_i \in S'$, $a_i \in L_H$ and $b_i \in L'_H$ and some μ_i for $i = 1, 2, \ldots$. Since (s, t) is a state that occurs infinitely often on π, there are infinitely many $\left(s_i, t_j\right) = (s, t)$. By Definition 5 and construction of π, there exist two paths in $\mathcal{A}$ and $\mathcal{B}$ respectively, such that

  $$\pi_{\mathcal{A}} = s_1\ \varrho_1\ a_1\ s_2\ \varrho_2\ a_2\ s_3\ldots$$
  $$\pi_{\mathcal{B}} = t_1\ \nu_1\ b_1\ t_2\ \nu_2\ b_2\ t_3\ldots$$

  Clearly $\pi_{\mathcal{A}}$ and $\pi_{\mathcal{B}}$ are divergent, since $a_i \in L_H$ and $b_i \in L'_H$ for $i = 1, 2, \ldots$. Since π is a fair path with respect to $\mathcal{A} \,||\, \mathcal{B}$ it immediately follows that $\pi_{\mathcal{A}}$ is a fair path with respect to $\mathcal{A}$ and $\pi_{\mathcal{B}}$ is a fair path with respect to $\mathcal{B}$. Since there are infinitely many $\left(s_i, t_j\right) = (s, t)$ we immediately know that there are infinitely many $s_i = s$ on $\pi_{\mathcal{A}}$ and infinitely many $t_j = t$ on $\pi_{\mathcal{B}}$. However, this means that $s \in d(\mathcal{A})$ and likewise $t \in d(\mathcal{B})$. Since both $\mathcal{A}$ and $\mathcal{B}$ are well-formed, we know that $\delta \in \mathit{enabled}(s)$ and $\delta \in \mathit{enabled}(t)$. Therefore $\delta \in \mathit{enabled}((s, t))$.
- Assume $\mathcal{A}$ carries out finitely many internal actions on π and $\mathcal{B}$ carries out infinitely many internal actions on π. Since π is infinite and the contribution of $\mathcal{A}$ to it is limited, there always exist π' and π'' such that all internal actions carried out by $\mathcal{A}$ are on π' and all internal actions carried out by $\mathcal{B}$ are on π''. Since $\mathcal{A}$ and $\mathcal{B}$ cannot synchronize on internal actions, it follows that π'' is of the form

  $$\pi'' = (s_1, t_1)\,\mu_1\, b_1\,(s_1, t_2)\,\mu_2\, b_2\,(s_1, t_3)\,\mu_3\, b_3\ldots,$$

  where $s_1 \in S$, $t_i \in S'$ and $b \in L'_H$ for some μ_1 for $i = 1, 2, \ldots$. Since π is divergent, obviously π'' is also divergent. Moreover if (s, t) occur infinitely often on π then they also occur infinitely often on π''. We must show that $\delta \in \mathit{enabled}(s)$ and $\delta \in \mathit{enabled}(t)$. We do so by showing $s \in q(\mathcal{A})$ and $t \in d(\mathcal{B})$. Since both are well-formed systems, the desired result immediately follows from **R1**. First we show that $s \in q(\mathcal{A})$. Since (s, t) occurs infinitely often on π it follows that without loss of generality $s_0 = s$, i.e. the finite progress of $\mathcal{A}$ can be neglected. Since π is a fair path, it follows that π'' is a fair path. Assume $\mathit{enabled}(s) \cap (L_O \cup L_H) \neq \emptyset$. That would mean that π'' does not execute some of its possible actions even though given the chance infinitely many times. This yields a contradiction, because π'' and consequently π would not be fair paths. We conclude $s \in q(\mathcal{A})$, because it does not enable output or internal actions. Next we show that $t \in d(\mathcal{B})$. By Definition 5 we have the following infinite path in $\mathcal{B}$:

  $$\pi_{\mathcal{B}} = t_1\ \nu_1\ b_1\ t_2\ \nu_2\ b_2\ t_3\ldots.$$

  Clearly, $\pi_{\mathcal{B}}$ is divergent, since $b_i \in L'_H$ for $i = 1, 2, \ldots$. Since π'' is fair and $s = s_0$ we know that $\pi_{\mathcal{B}}$ is fair. Furthermore, as we observed earlier (s, t) occurs infinitely often on π. Hence t occurs infinitely often on $\pi_{\mathcal{B}}$. Since $\pi_{\mathcal{B}}$ is divergent, fair and state finite, we conclude that $t \in d(\mathcal{B})$.
- This case is symmetric for the proof of the previous case.

2. To prove $\mathcal{A} \,||\, \mathcal{B}$ satisfies Rule **R2**, we must show that for all pairs $(s, t), (s', t') \in S''$:

   $$\text{If } (s, t) \xrightarrow{\mu,\delta}_{\mathcal{A}||\mathcal{B}} \left(s', t'\right) \text{ for some } \mu, \text{ then } \left(s', t'\right) \in q(\mathcal{A} \,||\, \mathcal{B}).$$

   From Definition 5 it follows that for this case we have $s \xrightarrow{\mu,\delta} s'$ and $t \xrightarrow{\nu\delta}$ for some μ, ν. Because $\mathcal{A}$ and $\mathcal{B}$ are well-formed and by Rule **R2**, we know that both s' and t' are quiescent. Thus, by Definition 5 we know that $(s', t') \in q(\mathcal{A} \,||\, \mathcal{B})$.
3. To prove that $\mathcal{A} \,||\, \mathcal{B}$ satisfies **R3** we must show for all pairs of states (s, t) and (s', t') in S'':

   $$\text{If } (s, t) \xrightarrow{\mu,\delta} \left(s', t'\right) \text{ for some } \mu, \text{ then } \mathit{Trd}\left(\left(s', t'\right)\right) \subseteq \mathit{Trd}((s, t)).$$

   Since $(s, t) \xrightarrow{\mu,\delta} (s', t')$ for some μ, we know that $s \xrightarrow{\nu_1,\delta} s'$ and $t \xrightarrow{\nu_2,\delta} t'$ for some ν_1, ν_2. Because $\mathcal{A}$ and $\mathcal{B}$ are both well-formed, we know that $\mathit{Trd}(s') \subseteq \mathit{Trd}(s)$ and $\mathit{Trd}(t') \subseteq \mathit{Trd}(t)$. Note that [STS13]

show that also $Traces^{<\omega}(s') \subseteq Traces^{<\omega}(s)$ and $Traces^{<\omega}(t') \subseteq Traces^{<\omega}(t)$. The proof arises now as a combination of this and Definition 5.

4. To prove that $\mathcal{A} \,||\, \mathcal{B}$ satisfies rule **R4** we must show that for all pairs of states $(s, t), (s', t'), (s'', t'') \in \mathcal{A} \,||\, \mathcal{B}$:

$$\text{If } (s, t) \xrightarrow{\mu,\delta}_{\mathcal{A}||\mathcal{B}} (s', t') \text{and} (s', t') \xrightarrow{\nu,\delta}_{\mathcal{A}||\mathcal{B}} (s'', t'') \text{ for some } \mu, \nu, \text{ then } Trd\left((s', t')\right) = Trd\left((s'', t'')\right).$$

Consider any pair of transitions of $(s, t) \xrightarrow{\mu,\delta}_{\mathcal{A}||\mathcal{B}} (s', t')$ and $(s', t') \xrightarrow{\nu,\delta}_{\mathcal{A}||\mathcal{B}} (s'', t'')$. By Definition 5 we know that then $s \xrightarrow{\mu_1,\delta}_{\mathcal{A}} s'$ and $s' \xrightarrow{\mu_2,\delta}_{\mathcal{A}} s''$ as well as $t \xrightarrow{\nu_1,\delta} t'$ and $t' \xrightarrow{\nu_2,\delta} t''$ for some $\mu_1, \mu_2, \nu_1, \nu_2$. To prove $Trd((s', t')) = Trd((s'', t''))$ we must show both the "$\subseteq$" and the "$\supseteq$" directions. Note that the latter directly follows from **R3** shown earlier. The proof of the first is similar to the techniques used in **R3** only using $Trd(s') = Trd(s'')$ and $Trd(t') = Trd(t'')$ instead of $Trd(s'') \subseteq Trd(s')$ and $Trd(t'') \subseteq Trd(t')$.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

[AK04] Al-Karaki JN, Kamal AE (2004) Routing techniques in wireless sensor networks: a survey. IEEE Wirel Commun 11(6):6–28

[BB04] Briones LB, Brinksma Ed (2004) A test generation framework for quiescent real-time systems. In: Proceedings of formal approaches to testing of software (4th international workshop), pp 71–85

[BB05] Bohnenkamp H, Belinfante A (2005) Timed testing with TorX. In: Formal methods Europe (FME), volume 3582 of LNCS, pp 173–188. Springer

[BB08] Bernardo M, Botta S (2008) A survey of modal logics characterising behavioural equivalences for non-deterministic and stochastic systems. Math Struct Comput Sci 18(01):29–55

[BD05] Beyer M, Dulz W (2005) Scenario-based statistical testing of quality of service requirements. In: Scenarios: models, transformations and tools, volume 3466 of LNCS, pp 152–173. Springer

[BDH+12] Bozga M, David A, Hartmanns A, Hermanns H, Larsen KG, Legay A, Tretmans J (2012) State-of-the-art tools and techniques for quantitative modeling and analysis of embedded systems. In: DATE, pp 370–375

[Bel10] Belinfante AEF (2010) JTorX: a tool for on-line model-driven test derivation and execution, volume 6015 of LNCS, pp 266–270. Springer

[BKL08] Baier C, Katoen J-P, Larsen KG (2008) Principles of model checking. MIT Press, Cambridge

[BNL13] Bernardo M, De Nicola R, Loreti M (2013) A uniform framework for modeling nondeterministic, probabilistic, stochastic, or mixed processes and their behavioral equivalences. Inf Comput 225:29–82

[Böh11] Böhr F (2011) Model based statistical testing of embedded systems. In: IEEE 4th international conference on software testing, verification and validation workshops (ICSTW), pp 18–25

[CDSMW09] Choi SG, Dachman-Soled D, Malkin T, Wee H (2009) Improved non-committing encryption with applications to adaptively secure protocols. In: ASIACRYPT, volume 5912 of LNCS, pp 287–302. Springer

[CDSY99] Cleaveland R, Dayar Z, Smolka SA, Yuen S (1999) Testing preorders for probabilistic processes. Inf Comput 154(2):93–148

[Coh80] Cohn DL (1980) Measure Theory. Birkhäuser, Boston

[CSV07] Cheung L, Stoelinga MIA, Vaandrager FW (2007) A testing scenario for probabilistic processes. J ACM 54(6):29

[DHvGM08] Deng Y, Hennessy M, van Glabbeek RJ, Morgan C (2008) Characterising testing preorders for finite probabilistic processes. CoRR

[DLT08] Desharnais J, Laviolette F, Tracol M (2008) Approximate analysis of probabilistic processes: logic, simulation and games. In: 5th international conference on quantitative evaluation of systems, pp 264–273

[Gog00] Goga N (2000) An optimization of the torx test generation algorithm. Xootic Mag 8(2):15–21

[GS15] Gerhold M, Stoelinga M (2015) Ioco theory for probabilistic automata. In: Proceedings of tenth workshop on MBT, pp 23–40

[GS16] Gerhold M, Stoelinga M (2016) Model-based testing of probabilistic systems, pp 251–268. Springer, Berlin

[GS17] Gerhold M, Stoelinga M (2017) Model-based testing of probabilistic systems with stochastic time. In: Proceedings of the 11th international conference on tests and proofs, TAP, LNCS. Springer **(to appear)**

[GSST90] van Glabbeek RJ, Smolka SA, Steffen B, Tofts CMN (1990) Reactive, generative, and stratified models of probabilistic processes, pp 130–141. IEEE Computer Society Press, Philadelphia

[Gui98] MATLAB Users Guide (1998) The Mathworks Inc., Natick, MA, vol 5, pp 333

[HC10] Hwang I, Cavalli AR (2010) Testing a probabilistic FSM using interval estimation. Comput Netw 54:1108–1125

[Her02] Hermanns H (2002) Interactive Markov chains: and the quest for quantified quality. Springer, Berlin

[HLM+08] Hessel A, Larsen KG, Mikucionis M, Nielsen B, Pettersson P, Skou A (2008) Testing real-time systems using UPPAAL, volume 4949 of LNCS, pp 77–117. Springer

[HM09] Hierons RM, Merayo MG (2009) Mutation testing from probabilistic and stochastic finite state machines. J Syst Softw 82:1804–1818
[HN10] Hierons RM, Núñez M (2010) Testing probabilistic distributed systems, volume 6117 of LNCS, pp 63–77. Springer
[HN12] Hierons RM, Núñez M (2012) Using schedulers to test probabilistic distributed systems. Formal Asp Comput 24(4–6):679–699
[HN17] Hierons RM, Núñez M (2017) Implementation relations and probabilistic schedulers in the distributed test architecture. J Syst Softw 132:319–335
[JDL02] Jeannet B, D'Argenio PR, Larsen KG (2002) Rapture: a tool for verifying Markov decision processes. In: Tools day
[JLS12] Jegourel C, Legay A, Sedwards S (2012) A platform for high performance statistical model checking—PLASMA. Springer, Heidelberg
[JS07] Jorgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans softw Eng 33(1):33–53
[KNP02] Kwiatkowska M, Norman G, Parker D (2002) PRISM: probabilistic symbolic model checker. In: computer performance evaluation: modelling techniques and tools, pp 200–204. Springer
[KY76] Knuth DE, Yao AC (1976) The complexity of nonuniform random number generation. In: Traub JF (ed) Algorithms and complexity: new directions and recent results. Academic Press, New York, pp 357–428
[LS89] Larsen KG, Skou A (1989) Bisimulation through probabilistic testing, pp 344–352. ACM Press, New York
[MBC+94] Marsan MA, Balbo G, Conte G, Donatelli S, Franceschinis G (1994) Modelling with generalized stochastic petri nets. Wiley, Hoboken
[Mil80] Milner R (1980) A calculus of communicating systems. Springer, New York
[NDG08] Nie J, Demmel J, Gu M (2008) Global minimization of rational functions and the nearest GCDs. J Glob Optim 40(4):697–718
[NH84] De Nicola R, Hennessy MCB (1984) Testing equivalences for processes. Theor Comput Sci 34(1):83–133
[Pfe11] Pfeffer A (2011) Practical probabilistic programming. In: Inductive logic programming, volume 6489 of LNCS, pp 2–3. Springer, Berlin
[PKB+14] Peters H, Knieke C, Brox O, Jauns-Seyfried S, Krämer M, Schulze A (2014) A test-driven approach for model-based development of powertrain functions. In: Agile processes in software engineering and extreme programming, volume 179 of LNBIP, pp 294–301. Springer
[Pro03] Prowell SJ (2003) Computations for Markov chain usage models. Technical Report
[Put14] Puterman ML (2014) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York
[PW14] Paige B, Wood F (2014) A compilation target for probabilistic programming languages. CoRR, arXiv:1403.0504
[RM85] Russell NJ, Moore RK (1985) Explicit modelling of state occupancy in hidden markov models for automatic speech recognition. In: Acoustics, speech, and signal processing. IEEE international conference on ICASSP'85, volume 10, pp 5–8
[RS14] Remke A, Stoelinga M (eds) (2014) Stochastic model checking. Rigorous dependability analysis using model checking techniques for stochastic systems—International Autumn School, ROCKS 2012, volume 8453 of LNCS. Springer
[Seg95] Segala R (1995) Modeling, verification of randomized distributed real-time systems. Ph.D. thesis, Cambridge, MA, USA
[Seg96] Segala R (1996) Testing probabilistic automata. In: CONCUR 96: concurrency theory, volume 1119, pp 299–314. Springer
[Sto02] Stoelinga MIA (2002) Alea jacta est: verification of probabilistic, real-time and parametric systems. Ph.D. thesis, Radboud University of Nijmegen
[STS13] Stokkink WGJ, Timmer M, Stoelinga MIA (2013) Divergent quiescent transistion sytems. In: Proceedings 7th conference on tests and proofs (TAP'13), LNCS
[SV99] Stoelinga M, Vaandrager F (1999) Root contention in IEEE 1394. In: Formal methods for real-time and probabilistic systems, volume 1601 of LNCS, pp 53–74. Springer, Berlin
[SVA04] Sen K, Viswanathan M, Agha G (2004) Statistical model checking of black-box probabilistic systems. In: Alur R, Peled D (eds) 16th conference on computer aided verification (CAV), pp 202–215
[SVA05] Sen K, Viswanathan M, Agha G (2005) On statistical model checking of stochastic systems. In: CAV, pp 266–280
[TBF05] Thrun S, Burgard W, Fox D (2005) Probabilistic robotics. MIT Press, Cambridge
[TBS11] Timmer M, Brinksma H, Stoelinga M (2011) Model-based testing. In: Software and systems safety: specification and verification, volume 30 of NATO science for peace and security, pp 1–32. IOS Press
[Tre96] Tretmans J (1996) Test generation with inputs, outputs and repetitive quiescence. Softw Concepts Tools 17(3):103–120
[Tre08] Tretmans J (2008) Model based testing with labelled transition systems. In: Formal methods and testing, volume 4949 of LNCS, pp 1–38. Springer
[vO06] van Osch M (2006) Hybrid input-output conformance and test generation. In: Proceeings of FATES/RV 2006, number 4262 in LNCS, pp 70–84
[WPT95] Walton GH, Poore JH, Trammell CJ (1995) Statistical testing of software based on a usage model. Softw Pract Exp 25(1):97–108
[WRT00] Whittaker JA, Rekab K, Thomason MG (2000) A Markov chain model for predicting the reliability of multi-build software. Inf Softw Technol 42(12):889–894

Received 18 November 2016
Accepted in revised form 26 August 2017 by Perdita Stevens, Andrzej Wasowski, and Ewen Denney
Published online 2 January 2018