Implementation relations and probabilistic schedulers in the distributed test architecture[☆]

Robert M. Hierons

Department of Computer Science, Brunel University London, United Kingdom

Manuel Núñez[*]

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain

Abstract

We present a complete framework to formally test systems with distributed ports where some choices are probabilistically quantified while other choices are non-deterministic. We define different *implementation relations*, that is, relations that state what it means for a system to be a valid implementation of a specification. We also study how these relate. In order to define these implementation relations we use *probabilistic schedulers*, a more powerful version, including probabilistic choices, of a notion of scheduler introduced in our previous work. Probabilistic schedulers, when applied to either a specification or an implementation, resolve all the possible non-determinism, so that we can compare purely probabilistic systems.

Keywords: Model-based testing; probabilistic systems; distributed systems.

1. Introduction

Software Testing is the most widely used technique to validate the correctness of software systems. Classically, testing has been a manual and labour-intense activity, taking more than 50% of the budget of software projects [1]. However, in the last two decades there has been a significant line of work that *formalises* the area. There are now comprehensive studies of the field [2, 3, 4], tools to support formal approaches to testing [5, 6], and significant evidence of industrial uptake [7, 8, 9, 10, 11]. In formal testing it is important to use a suitable *implementation relation* that encodes the notion of conformance used; otherwise

[☆]Research partially supported by the MINECO/FEDER project DArDOS (TIN2015-65845-C3-1-R) and the Comunidad de Madrid project SICOMORo-CM (S2013/ICE-3006).

[*]Corresponding author

Email addresses: `rob.hierons@brunel.ac.uk` (Robert M. Hierons), `mn@sip.ucm.es` (Manuel Núñez) testing may be unsound and inefficient. Essentially, an implementation relation defines what it means for a *System Under Test (SUT)* to be a valid implementation of a given specification. The de facto standard implementation relation is **ioco** [12].

If the SUT has multiple physically distributed interfaces at which it interacts with its environment then one might place a separate local tester at each interface (port). If the local testers do not synchronise their actions and there is no global clock then one is testing in the distributed test architecture [13]. Although the initial work on the distributed test architecture concentrated on observability and controllability issues (see the next section), later work extended and adapted existing implementation relations to the distributed framework. The **dioco** relation [14, 15] follows the pattern of the classical **ioco** relation: the SUT should not produce an unexpected output as the reaction to a sequence of actions allowed by the specification. However, its definition, and the study of its properties, has to take into account the fact that different sequences of actions must be identified as equivalent because they cannot be distinguished in distributed testing; they look identical to the local testers placed at the ports of the SUT. In our previous work [16, 17] we studied the impact of the inclusion of probabilistic information into our systems. We discovered that non-determinism means that one cannot simply compare the probabilities of equivalence classes of traces of the SUT and specification and so we initially studied a class of models where this non-determinism does not cause problems [16]. Later we generalised this work by using schedulers to resolve non-deterministic choices in processes. Essentially, schedulers can be seen as being models of the environment. However, our previous work used a restricted notion of a scheduler, which did not include probabilistic information. It is natural to ask whether the introduction of probabilities into schedulers might affect the implementation relations and associated analysis.

This paper introduces a new (probabilistic) type of scheduler. Based on this, we define three new implementation relations for distributed testing of probabilistic input-output transition systems (PIOTSs). We also analyse how these new implementation relations are related to one another and to our former implementation relations. It transpires that the strongest implementation relation is equivalent to the strongest of our previously defined implementation relations but that the weaker implementation relations are affected by the use of probabilistic schedulers.

The paper is structured as follows. Section 2 describes related work. Section 3 describes the types of models used in this paper. Section 4 describes deterministic schedulers and implementation relations defined in terms of such schedulers. Section 5 defines probabilistic schedulers and how they interact with PIOTSs. Section 6 defines new implementation relations in terms of probabilistic schedulers and compares these with previously defined implementation relations. Finally, in Section 7 we discuss the results and possible lines of future work.

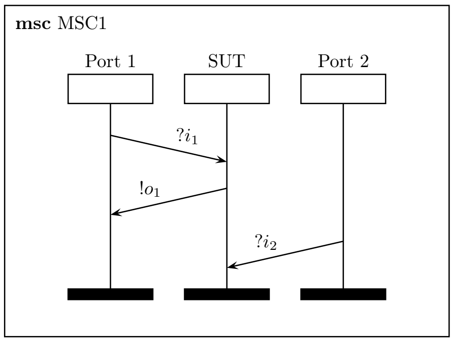

Figure 1: A test case with a controllability problem

2. Related work

In this section we review previous work on the two main lines that we consider in this paper: testing in the distributed architecture and testing probabilistic systems.

2.1. Distributed testing

The initial work on distributed testing was developed in the context of testing the implementation of a communications protocol based on a finite state machine (FSM) model [18, 19, 20]. Here, one local tester acted as the layer above the implementation being tested and the other local tester resided on a different machine. These local testers acted independently and there was no global clock and so testing was in the distributed test architecture. This initial work identified two additional practical issues introduced by distributed testing and called these controllability and observability problems. A *controllability problem* occurs if the local tester at port p cannot determine when to send an input since this tester can only make decisions on the basis of the observations at port p. Let us suppose, for example, that a test case starts with input $?i_1$ at port 1, this should lead to output $!o_1$ at port 1 only, and the tester at port 2 should then send input $?i_2$ (see Figure 1). The problem is that the tester at port 2 cannot know when to send its input since it did not observe either $?i_1$ or $!o_1$. A test sequence that has no controllability problems is said to be *controllable.* In contrast, *observability problems* refer to the fact that two traces (sequences of inputs and outputs) might be different despite no local tester observing this difference. Let us suppose, for example, that the specification allows the trace $?i_1!o_2?i_2!o_1!o_2$ in which input $?i_1$ is at port 1, input $?i_2$ is at port 2, output $!o_1$ is at port 1, and output $!o_2$ is at port 2. If the SUT instead produces trace $?i_1!o_1!o_2?i_2!o_2$ then the expected trace has not occurred but the local testers make the expected observations: in both cases the tester at port 1 observes $?i_1!o_1$ and the tester at port 2 observes $!o_2?i_2!o_2$. This is illustrated in Figure 2. As a result, there has been much interest in the development of FSM-based test

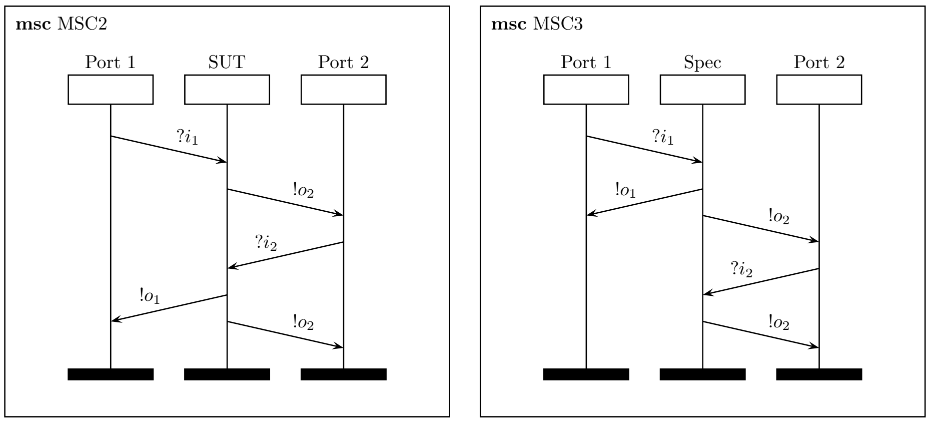

Figure 2: Observationally equivalent traces

generation techniques that return test sequences that are free from controllability and/or observability problems (see, for example, [21, 22, 23, 24, 25, 26, 27]). It has also been noted that if the local testers can be connected using an external network then message exchange between the local testers can sometimes be used to overcome controllability and observability problems [28, 29]. However, the introduction of such a network can increase the cost of testing and it may not be possible to execute some test sequences if there are timing constraints [30].

It is not difficult to show that test techniques that only return controllable test sequences must have limitations. For example, one can construct an FSM where there are states that cannot be reached using controllable test cases. This observation led to two additional lines of research. One line of research explored test generation without restricting attention to controllable test cases, with it being found that many classic problems, for testing from an FSM, become undecidable [31]. A second line of research aimed to capture the power of distributed testing and so defined corresponding implementation relations. The initial work was in the context of testing from an FSM [32]. This was then extended to define the **dioco** implementation relation for distributed testing from an IOTS [14, 15].

Almost all work on distributed testing has used either FSM or IOTS models. In such models, transitions are labelled by either individual actions or sequences of actions and so behaviours are sequences of inputs and outputs[1]. This corresponds to an interleaving-based approach to concurrency. In contrast, there has been some work in which a behaviour is a partially ordered multi-set of actions [33, 34, 35, 36]. While such an approach can lead to more compact models, the current work does not take into account the distributed nature of testing and

[1]An output might be a tuple of values, with up to one value at each port

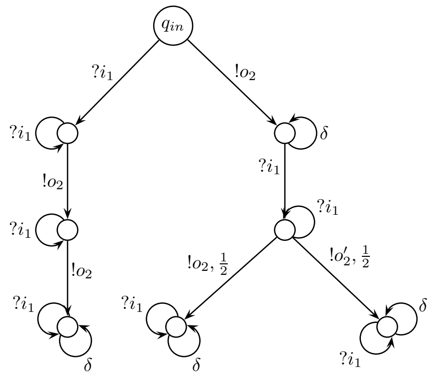

Figure 3: Races between inputs and outputs at different ports

additional mechanisms, such as using time to label events, are needed [37].

2.2. Testing of probabilistic systems

The study of (formal) probabilistic testing frameworks can be dated back to the end of the 1980s. This is already a well established area, with many contributions extending classical formalisms (Process Algebras, I/O Automata, Finite State Machines, Input Output Transition Systems, among others) to include probabilistic information [38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52].

One interesting issue is how probabilities governing the behaviour of systems are extracted during testing. Usually, probabilistic testing frameworks assume that the two systems being compared are *visible* (that is, we have their formal representation). Therefore, it is possible to compare probabilities because we can *see* them. In practice, this approach is not feasible in testing because we have access to the probabilities of one of the systems (the specification) but if the SUT is a black box we do not have access to its probabilities. In this case, the tester must try to estimate these probabilities by performing experiments [44]. For example, the tester can apply an input n times and this will induce a probability distribution on the observed outputs. Then, a contrast hypothesis can decide whether it is likely that the experimental probabilities are generated by the probability distribution extracted from the specification (note that we can compute the latter because we have access to the specification).

The previously mentioned approaches considered the testing of an isolated, non-distributed system. In our previous work [16] we presented the first, to the best of our knowledge, formal testing framework where probabilities were included in the specification and analysis of systems with distributed ports. We considered, and we use the same formalism in this paper, labelled transition systems where we distinguish between inputs and outputs, there are multiple ports, and each action occurs at a particular port. In order to model probabilistic information, we use a combination of the *reactive* and *generative* approaches. Our model is reactive for inputs: given state s and input $?i$, the sum of all the probabilities of the transitions leaving s with input $?i$ is 1. However, it is generative for outputs: given state s, the sum of the probabilities of the transitions leaving s and labelled by an output is 1. Note that this combination of the two models is not original and has been widely used in the literature. The interested reader is referred to previous work [53, 54] for longer explanations on the appropriate use of the reactive and generative models and for motivations on the usefulness of a mixed reactive-generative model.

One problem that we confronted in the original formulation of our framework for testing probabilistic systems in the distributed architecture was that we can have races between events at different ports. Even though our reactive-generative approach can be used to model systems with inputs and outputs, in this case the setting does not provide probabilistic information regarding the outcome of such races. For example, consider the system depicted in Figure 3, where we have omitted probabilities equal to 1 and δ denotes quiescence (quiescence is not relevant for this example). In addition, indexes indicate the port (1 or 2) at which the action is being observed. In this case, it is not possible to compute the probability that we should associate with global traces that are indistinguishable from $?i_1!o_2!o_2$ (we have two of them: $?i_1!o_2!o_2$ and $!o_2?i_1!o_2$) because whether we observe $?i_1!o_2!o_2$ depends on the outcome of a race between $?i_1$ and $!o_2$ in the initial state and we have no probabilistic information regarding this race. Actually, if we are not careful then we might end up assigning *probability* 1.5 to this set of traces. In order to partially overcome this problem we might probabilistically quantify the choice between performing actions at different ports. However, if we did this then we would have to probabilistically quantify the choice between different inputs at a certain port, the relative probability of performing inputs at different ports, and quantify the probability between inputs and outputs. Unfortunately, this would lead to complicated models and it is unclear how realistic it is to assume that one can know whether two inputs are simultaneously offered at two (distant) ports. As a result, we initially outlawed these types of races and provided a condition under which such races cannot occur [16]. A second step was to consider a basic notion of *schedulers*, also called *adversaries* in the literature, in our framework [17]. Schedulers can be used to resolve races and so their use allows an additional degree of nondeterminism in the description of systems and we do not have to forbid races between events at different ports. In this paper we consider schedulers that can include probabilistic choices between different alternatives. We study the impact of these new *probabilistic* schedulers (our previous schedulers are a particular case of the new ones) in the definition of new versions of our semantic notions.

Our implementation relations are defined in terms of the traces, actually, the sets of *indistinguishable* traces, that can be observed at different ports. Therefore, our methodology has some similarities with work on testing of probabilistic automata [55, 56, 57, 40]. In particular, adversaries are used to extract sequences from probabilistic systems [57]. However, the assumption of distributed ports, the distinction between inputs and outputs and the use of schedulers in the way that we do it is not considered in that line of work. Some of the ideas underlying our notion of schedulers are similar to those found in previous work [58] but we use a different formalism (a unique system with distributed ports versus the parallel composition of different systems) and the main goal of our research is different (we concentrate on implementation relations).

3. Preliminaries

In this section we start by describing the formalisation we use for systems (probabilistic labelled transition system and probabilistic input output transition systems). First we define some notation.

We are concerned with the behaviour of state-based systems and such a behaviour will be denoted by a sequence of actions. Throughout this paper we will let ϵ denote the empty sequence. We will also use multi-sets of probabilities, with $\{\!\!\{ p_1, p_2, \ldots, p_k \}\!\!\}$ denoting such a multi-set. Finally, since we will consider systems with different ports, we denote by $\mathcal{O}$ the (finite) set of ports. Throughout the paper we assume that this set is fixed.

The two tables appearing in Figures 4 and 5 summarise the main concepts and implementation relations that we will use in this paper. The first table describes definitions of systems and notions while the second table gives an intuitive description, although not as precise as the actual definition, of the different implementation relations that we use in the paper.

3.1. Probabilistic labelled transition systems

The basic type of model used is a probabilistic labelled transition system. We include a special action, δ, and require that δ is a possible action whenever no other actions can occur. We include δ in the definition since, once we consider models with input and output, we will use δ to denote quiescence: the model being in a state where it cannot produce output or change state without first receiving input. Therefore, the interaction between systems and schedulers can produce this type of transitions.

Definition 1 *A* probabilistic labelled transition system (PLTS) *s is defined by a tuple $(Q, \mathcal{A}ct, T, q_{in})$ in which Q is a countable set of states, $q_{in} \in Q$ is the initial state, $\mathcal{A}ct$ is a countable set of actions, with $\delta \in \mathcal{A}ct$, and $T \subseteq Q \times \mathcal{A}ct \times Q \times (0, 1]$ is the transition relation. A transition (q, a, q', p) means that when in state q, with probability p the next event moves s to state q' with action $a \in \mathcal{A}ct$. We cannot have two transitions $(q, a, q', p) \in T$ and $(q, a, q', p') \in T$ in which $p \neq p'$. We say that transition (q, a, q', p) has start state q, end state q' and label a. We require that for every state $q \in Q$ either $\sum \{\!\!\{ p \mid \exists a, q' : (q, a, q', p) \in T \}\!\!\}$ is equal to 1 or q is a deadlock state and so this sum is equal to zero. We extend the set of transitions T to a new set T_δ by adding the transition $(q, \delta, q, 1)$ for each deadlock state q.*

Let $\mathcal{O} = \{1 \ldots m\}$ be the set of ports. For port $o \in \mathcal{O}$ we let $\mathcal{A}ct_o$ denote the set of actions that can be observed at o. Thus, for all $o \in \mathcal{O}$ we have that $\delta \in \mathcal{A}ct_o$ and also that $\mathcal{A}ct_1 \setminus \{\delta\}, \ldots, \mathcal{A}ct_m \setminus \{\delta\}$ partition $\mathcal{A}ct \setminus \{\delta\}$. We let $\mathcal{PLTS}(\mathcal{A}ct)$ denote the set of PLTSs with action set $\mathcal{A}ct$.

Types of systems		
Notation	*Def.*	*Explanation*
PLTS	Def. 1	Labelled transition systems with a unique probability distribution for all the actions departing a given state.
PIOTS	Def. 6	Labelled transition systems with a distinction between *reactive* inputs (a probability distribution for each of the inputs departing a given state) and *generative* outputs (a unique probability distribution for all the outputs departing a given state).
Main notions and concepts		
$\sim$ and $[\sigma]$	Def. 3	$\sim$ is an equivalence relation between traces: two traces are related if all their local projections are equal. $[\sigma]$ is the equivalence class of σ.
prob	Def. 2, 4, 15 and 16	This function is overloaded. It computes probabilities of traces, set of traces, equivalence classes, etc.
deterministic scheduler	Def. 8	System that resolves non-determinism in PIOTSs: when applied to a PIOTS it returns a PLTS.
probabilistic scheduler	Def. 13	Same purpose as deterministic schedulers but allowing probabilistic choices between alternatives. They are applied only to PIOTSs.
path and $pre(\rho)$	Def. 15	Sequence of consecutive transitions of a PLTS/scheduler. $pre(\rho)$ is the set of prefixes of the path ρ.

Figure 4: Main concepts

A state $q \in Q$ defines a PLTS derived from s by setting the initial state to q. As a result, in an abuse of notation we consider q to be the PLTS $(Q, \mathcal{A}ct, T, q)$ with unreachable states (and corresponding transitions) removed.

We make a number of restrictions to the PLTSs that we consider. A PLTS s is *connected* if for each state q of s there is a sequence

$$(q_{in}, a_1, q_2, p_1)(q_2, a_2, q_3, p_2) \dots (q_k, a_k, q, p_k) \in T^*$$

We restrict attention to connected PLTSs; this is not problematic since, if a PLTS is not connected, it is possible to remove the unreachable states without affecting the behaviour. We assume that all transitions have non-zero probability; if this condition does not hold then we can simply delete the transitions that have zero probability. We also do not allow two different transitions to have the same starting state, ending state and label (and so to be of the form (q, a, q', p) and (q, a, q', p')); such a pair of transitions is equivalent to having the transition $(q, a, q', p + p')$. In the graphical representation of a system we omit the names

Implementation relations		
Notation	*Def.*	*Explanation*
$r \sqsubseteq s$	Def. 5	A PLTS r is correct with respect to another PLTS s if for all trace $\sigma\delta$ of s both processes return the same probability for $[\sigma\delta]$ (δ denotes quiescence).
$r \equiv s$	Def. 5	When restricted to a *finitary* class of processes the previous relation is an equivalence.
$r \equiv_d^s s$	Def. 11	A PIOTS r is correct with respect to another PIOTS s if for all deterministic scheduler, its application to each of the processes return PLTSs that are equivalent under $\equiv$. If we apply a scheduler to the SUT then the same scheduler must provide an equivalent process when applied to the specification.
$r \sqsubseteq_d^w s$	Def. 12	A PIOTS r is correct with respect to another PIOTS s if for all deterministic scheduler $\mathcal{G}_r$ there exists a deterministic scheduler $\mathcal{G}_s$ such that the application of $\mathcal{G}_r$ to r and the application of $\mathcal{G}_s$ to s return PLTSs that are equivalent under $\equiv$. If we apply a scheduler to the SUT then it is sufficient that there is a (possibly different) scheduler that leads to an equivalent process.
$r \equiv_p^s s$	Def. 17	Similar to $\equiv_d^s$ but for probabilistic schedulers.
$r \sqsubseteq_p^w s$	Def. 18	Similar to $\sqsubseteq_d^w$ but for probabilistic schedulers.
$r \equiv_p^w s$	Def. 19	Largest equivalence contained in $\sqsubseteq_p^w$.

Figure 5: Implementation relations

of irrelevant states and probabilities that are equal to 1. Moreover, a transition labelled by a set of actions is shorthand for a set of different transitions: one for each action. We will be interested in PLTSs that are formed from the composition of a probabilistic input-output transition system and a scheduler and it will transpire that, in such PLTSs, all δ transitions have probability 1. In Figure 6 we present three PLTSs with two ports and the following sets of actions: $\mathcal{A}ct_1 = \{\alpha, \beta\}$ and $\mathcal{A}ct_2 = \{\lambda, \mu\}$.

We use a generative interpretation of probabilities under which for each state s of the system, the probabilities of the transitions leaving s sum to 1 if δ self-loops are added to deadlocked states. The following shows how one can assign probabilities to sequences of actions (traces).

Definition 2 *Given a PLTS $s = (Q, \mathcal{A}ct, T, q_{in})$, state $q \in Q$, and $\sigma \in \mathcal{A}ct^*$, we let $prob(q, \sigma)$ denote the probability of performing the sequence σ from state q. Formally,*

$$prob(q,\sigma) = \begin{cases} 1 & \text{if } \sigma = \epsilon \\ \sum \{\!\mid p \cdot prob(q',\sigma') \mid (q,a,q',p) \in T \mid\!\} & \text{if } \sigma = a\sigma' \end{cases}$$

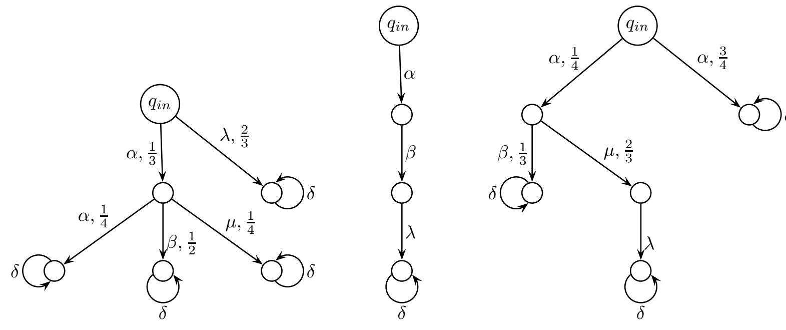

Figure 6: Examples of PLTSs.

A sequence $\sigma \in \mathcal{A}ct^*$ is said to be a trace *of s if $prob(q_{in}, \sigma) > 0$. We let $L(s)$ denote the set of traces of PLTS s.*

In distributed testing we have a set of physically distributed ports at which the system interacts with its environment. An event $a \in \mathcal{A}ct \setminus \{\delta\}$ is observed at a particular port: the port o such that $a \in \mathcal{A}ct_o \setminus \{\delta\}$. There is a separate tester at each port and the tester at port o only observes the events that occur at o (those in $\mathcal{A}ct_o$). As a result of this we have a notion of observational equivalence where two traces are equivalent if they have the same set of projections at the ports.

Definition 3 *Let $\sigma \in \mathcal{A}ct^*$ and $o \in \mathcal{O}$. The projection of the trace σ on the port o, denoted by $\pi_o(\sigma)$, is the sequence of events observed at port o if the trace σ occurs.*

$$\pi_o(\sigma) = \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ \pi_o(\sigma') & \textit{if } \sigma = a\sigma' \textit{ and } a \notin \mathcal{A}ct_o \\ a\pi_o(\sigma') & \textit{if } \sigma = a\sigma' \textit{ and } a \in \mathcal{A}ct_o \end{cases}$$

Overloading the previous notation, we let $\pi_o(C) = \{\pi_o(\sigma) | \sigma \in C\}$ for each $C \subseteq \mathcal{A}ct^*$.

Given two traces $\sigma, \sigma' \in \mathcal{A}ct^*$ we say that $\sigma \sim \sigma'$ if and only if $\pi_o(\sigma) = \pi_o(\sigma')$ for all $o \in \mathcal{O}$. Clearly $\sim$ is an equivalence relation and it captures the situation in which no port can distinguish between σ and σ'. We denote by $[\sigma]$ the equivalence class of σ.

Note that all testers observe δ because, as we have previously stated, $\delta \in \mathcal{A}ct_o$ for all $o \in \mathcal{O}$. In practice, quiescence (δ) is detected through a timeout. If we use a sufficiently large time before a next input is supplied then all of the local testers will observe quiescence. Later on we will introduce an invisible action τ, which will not be observed at any port. It will transpire that the notion of equivalence can be directly applied when we consider sequences containing invisible actions; we simply compare the traces formed by deleting invisible actions.

Example 1 *Consider the trace $\sigma = a_1 b_2 c_1 \delta d_1 \delta$, where the index denotes the port where the action is observed. The following traces belong to $[\sigma]$*

$$b_2 a_1 c_1 \delta d_1 \delta \qquad a_1 c_1 b_2 \delta d_1 \delta \qquad b_2 \tau a_1 c_1 \delta d_1 \delta \qquad a_1 \tau c_1 b_2 \delta \tau d_1 \delta$$

On the contrary, the following traces do not belong to $[\sigma]$

$$b_2 c_1 a_1 \delta d_1 \delta \qquad a_1 c_1 \delta b_2 d_1 \delta \qquad a_1 c_1 b_2 d_1 \delta \delta$$

The problem with these traces is that the order between actions is not preserved at a certain port. In the first case, the problem happens at port 1 (a_1 and c_1 are swapped); in the second case, the problem happens at port 2 (b_2 and δ are swapped) while the last case presents problems at both ports.

This paper concerns implementation relations, which are defined in terms of observations that can be made of the implementation and specification. Since we cannot distinguish between two traces that are equivalent under $\sim$, an observation will be an equivalence class of $\sim$. As a result, we consider the probability of equivalence classes of traces rather than single traces.

Definition 4 *Let $s = (Q, \mathcal{A}ct, T, q_{in})$ be a PLTS and $\sigma \in \mathcal{A}ct^*$. We define the* probability with which s performs the equivalence class $[\sigma]$, *denoted by $prob(s, [\sigma])$, as*

$$\sum \{\!| \; prob(q_{in}, \sigma') \mid \sigma' \in [\sigma] \; |\!\}$$

In testing, we will not directly compare traces of processes but instead we will compare projections of traces. We therefore wish to know that the projections observed are all projections of the same trace of a process. The approach used is to restrict observations to traces that end in δ; states that are said to be quiescent. The idea is that if we reach a state where δ can be produced then we know that the system cannot immediately produce an output and, therefore, we can study the observed projections at each port. This allows us to know that we are combining projections of the same global trace. For example, if we do not apply this restriction then we have that a system r that can just perform the trace $?i_1!o_2'!o_1$ would not be a good implementation of a specification s that can only perform the trace $?i_1!o_1!o_2'$: the probabilities associated with $[?i_1!o_1]$ are, respectively, equal to 0 and 1. The above observations lead to the following implementation relation [16].

Definition 5 *Let s, r be PLTSs with the same set $\mathcal{A}ct$ of actions. We write $r \sqsubseteq s$ if for every $\sigma \in \mathcal{A}ct^*$ such that $\sigma\delta$ is a trace of s, we have that $prob(s, [\sigma\delta]) = prob(r, [\sigma\delta])$.*

It is straightforward to show that if the number of traces of r and s concluding with a single occurrence of δ are both finite then $r \sqsubseteq s$ holds if and only if $s \sqsubseteq r$ [17]. To see this, let us suppose that $r \sqsubseteq s$ and $[\sigma_1\delta], \ldots, [\sigma_k\delta]$ are the equivalence classes of traces of s that end in a single δ. Then the sum of the probabilities of these equivalence classes, in s, is 1. Since $r \sqsubseteq s$, these equivalence classes have the same probabilities in r and s and so r cannot have any traces, that end in a single δ, that are not in the equivalence classes $[\sigma_1\delta], \ldots, [\sigma_k\delta]$. The fact that $s \sqsubseteq r$ follows from this and the equivalence classes having the same probabilities in r and s. Thus, for processes with finite sets of traces we have that $\sqsubseteq$ is an equivalence relation and so at times we will use the symbol $\equiv$. Note that this is not true in general; if a process s does not have finite sets of traces then the above argument need not hold since, for instance, s might have a trace that cannot be extended to a trace of s that ends in δ. For example, let s be a PLTS with a unique state and a self-loop transition labelled by an action a with probability 1. It is obvious that for all PLTS r we have $r \sqsubseteq s$, since s does not have traces ending with quiescence, but the reverse relation, that is, $s \sqsubseteq r$, does not necessarily hold. Although the previous relation has obvious shortcomings, for instance, it will identify systems with different *non-quiescent* infinite traces, it will be used to define other relations.

3.2. Probabilistic input output transition systems

While PLTSs allow us to describe probabilistic systems, they do not distinguish between inputs and outputs. However, in the context of testing it is often important to make this distinction since inputs and outputs play different roles; the testers control the application of inputs and the system controls the generation of outputs. This has led to models that distinguish between inputs and outputs. We now define a probabilistic IOTS that has multiple ports [16, 17].

In defining a model that has inputs and outputs and also probabilities, one has to decide how to assign probabilities. Since the environment, or tester, controls inputs it makes little sense to define the probability of an input when producing a model of the system. Thus, we have a *reactive* scenario for inputs. In contrast, since the system controls outputs we have a *generative* scenario for outputs. We do attach probabilities to transitions with inputs since for a state q and input $?i$ there may be more than one transition leaving q with input $?i$: the environment chooses the input but the system determines which transition to take. During the rest of the paper, if we are dealing with systems where there is a distinction between inputs and outputs we let $\mathcal{A}ct = \mathtt{In} \cup \mathtt{Out} \cup \{\delta\}$ denote the complete set of actions that can be performed by systems.

Definition 6 *A* probabilistic input-output transition system (PIOTS) *s is defined by a tuple $(Q, \mathtt{In}, \mathtt{Out}, T, q_{in})$ in which Q is a countable set of states, $q_{in} \in Q$ is the initial state, $\mathtt{In}$ is a countable set of inputs, $\mathtt{Out}$ is a countable set of outputs, and $T \subseteq Q \times (\mathtt{In} \cup \mathtt{Out}) \times Q \times (0, 1]$ is the transition relation. A transition (q, a, q', p) means that from state q it is possible to move to state q' with action $a \in \mathtt{In} \cup \mathtt{Out}$ with probability p. Again, we cannot have two transitions $(q, a, q', p), (q, a, q', p') \in T$ in which $p \neq p'$. If $a \in \mathtt{Out}$ then we should interpret the probability p of (q, a, q', p) as meaning that if an output occurs in state q before input is provided*[2] *then with probability p this transition occurs. Therefore,*

[2]We do not quantify time in our models. However, if the system under test is in a state

for every state q we must have that $\sum \{\!| p \mid \exists q', a : (q, a, q', p) \in T \wedge a \in \mathtt{Out} |\!\}$ is either 1 or 0 (if the state cannot produce any output). Further, if $a \in \mathtt{In}$ then we must have that the sum of the probabilities of transitions leaving q with input a, that is $\sum \{\!| p \mid \exists q' : (q, a, q', p) \in T |\!\}$, is 1. This means that once an input a is chosen by the environment, we can forget the other available inputs and concentrate on the probability distribution function governing the transitions labelled by a.

A state $q \in Q$ is quiescent *if there are no outgoing transitions from q labelled by output. We extend the set of transitions T to a new set T_δ by adding the transition $(q, \delta, q, 1)$ for each quiescent state q.*

We partition the set $\mathtt{In}$ of inputs into $\mathtt{In}_1, \ldots, \mathtt{In}_m$ in which for port $o \in \mathcal{O}$ we have that $\mathtt{In}_o$ is the set of inputs that can be received at port o. Similarly, we partition the set $\mathtt{Out}$ of outputs into sets $\mathtt{Out}_1, \ldots, \mathtt{Out}_m$. We let $\mathcal{P}IOTS(\mathtt{In}, \mathtt{Out})$ denote the set of PIOTSs with input set $\mathtt{In}$ and output set $\mathtt{Out}$. Given port o we let $\mathcal{A}ct_o = \mathtt{In}_o \cup \mathtt{Out}_o \cup \{\delta\}$ denote the set of events that can be observed at o.

We say that s is input-enabled *if for all $q \in Q$ and $a \in \mathtt{In}$ there is some $q' \in Q$ and probability p such that $(q, a, q', p) \in T$. We say that s is* output-divergent *if it can reach a state from which there is an infinite sequence of transitions whose label contains outputs only.*

In the context of testing it is normal to assume that the implementation is input-enabled; it cannot block input from the environment. In this paper we also assume that specifications are input-enabled since this simplifies the exposition. Therefore, since all our systems (specifications and implementations) are input-enabled, we restrict ourselves to consider PIOTSs that are input-enabled. This is why we require that for a state q and input $?i$ the sum of probabilities of transitions from q with input $?i$ is 1. We also require that processes, specifications and implementations, are not output-divergent. Intuitively, a PIOTS can either process a received input or produce a certain output (in particular, if the system is in a state where no outputs can be produced, it can produce δ).

Similar to PLTSs, we assume that all PIOTSs considered are connected. If one or more outputs are possible from a state q then the sum of the probabilities of the transitions from q with outputs is 1; if no outputs are possible from q then there is a self-loop transition from q with label δ and probability 1. As a result, the sum of the probabilities associated with events from $\mathtt{Out} \cup \{\delta\}$ in a state q is always equal to 1; for all $q \in Q$ we have $\sum \{\!| p \mid \exists q', a : (q, a, q', p) \in T_\delta \wedge a \notin \mathtt{In} |\!\} = 1$. As usual, we precede the name of an input by ? and we precede the name of an output by !. The name of an input or output at port o will often have o as a subscript. For example, $?i_1$ will normally denote an input at port 1 and $!o_2$ will normally denote an output at port 2. In Figure 7 we present a PIOTS with two ports and the following sets of actions: $\mathcal{A}ct_1 = \{?i_1, !o_1\}$ and $\mathcal{A}ct_2 = \{?i'_2, !o'_2, !o''_2\}$.

where it can produce output and the environment/tester is in a state where it can provide input then there is a race that, in practice, will be resolved based on whether the system under tester or environment acts first.

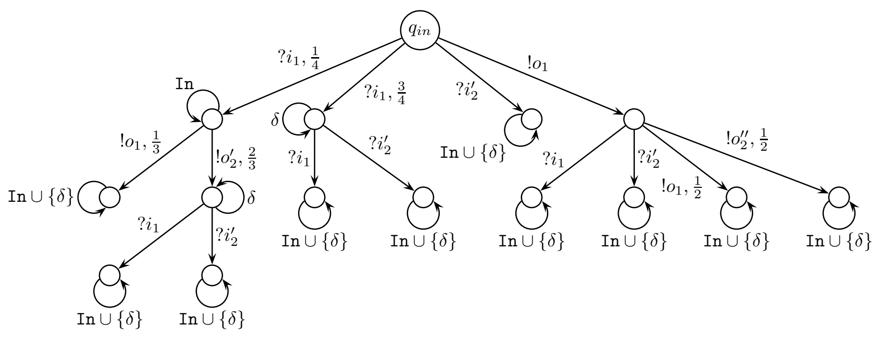

Figure 7: Example of PIOTS

Global traces are sequences of actions from $\mathcal{A}ct$ and so can contain quiescence.

Definition 7 *Given a PIOTS $s = (Q, \mathtt{In}, \mathtt{Out}, T, q_{in}) \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, we use the following notation.*

1. *If $(q, a, q', p) \in T_\delta$, for some probability $0 < p \leq 1$ and action $a \in \mathcal{A}ct$, then we write $q \xrightarrow{a} q'$ meaning that s can perform a from state q and reach, afterwards, the state q'. We also write $q \xrightarrow{a}$ meaning that s can perform a from state q.*
2. *We write $q \overset{\sigma}{\Longrightarrow} q'$ for $\sigma = a_1 \ldots a_m \in \mathcal{A}ct^*$ if there exist $q_0, \ldots, q_m \in Q$, with $q = q_0$ and $q' = q_m$, such that for all $0 \leq i < m$ we have that $q_i \xrightarrow{a_{i+1}} q_{i+1}$.*
3. *If there exists q' such that $q_{in} \overset{\sigma}{\Longrightarrow} q'$ then we say that σ is a* trace *of s and we write $s \overset{\sigma}{\Longrightarrow}$. We let $L(s)$ denote the set of traces of s. A trace σ of s is said to be a* quiescent trace *if $q_{in} \overset{\sigma}{\Longrightarrow} q'$ for a quiescent state q'.*

Although traces do not include probabilistic information, we will include such information when defining implementation relations. As explained in Section 2, we require schedulers in order to reason about distributed testing of PIOTSs.

4. Deterministic schedulers

In this section we outline the previously defined notion of deterministic schedulers, which have been called *global schedulers* [17], and how such a scheduler interacts with a system. We use the term deterministic to emphasise the difference between these and the probabilistic schedulers introduced in Section 5. We then explain how implementation relations can be defined in terms of deterministic schedulers. The essential idea is that a deterministic scheduler defines a possible environment; their use allows us to reason about how a PIOTS might behave with different environments. In the next section we will define a more general type of scheduler that quantifies the probabilities of actions performed by the environment.

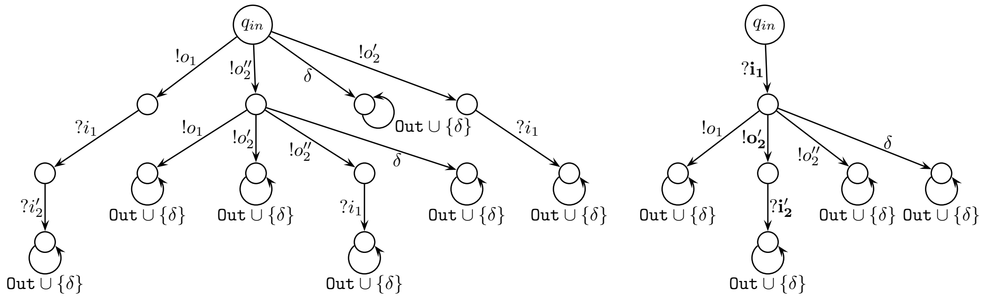

Figure 8: Deterministic schedulers

Definition 8 *Let* `In` *and* `Out` *be sets of inputs and outputs, respectively. A* deterministic scheduler *for* `In` *and* `Out` *is a tuple* $\mathcal{G} = (Q', \mathtt{In}, \mathtt{Out}, T', q'_{in})$ *such that* Q' *is a finite set of states, with* $q'_{in} \in Q'$ *its initial state, and* $T' \subseteq Q' \times \mathcal{A}ct \times Q'$ *its transition relation that satisfies the following:*

- *The graph having vertex set* Q' *and edges set* T' *is a tree with the exception of its* leaves*, which will have self-loop transitions labelled by each of the outputs belonging to* `Out` *and by* δ*.*
- *For all* $q \in Q'$*, one of the following possibilities holds:*
  - *There exists one* $a \in \mathtt{In}$ *and* $q' \neq q$ *such that* $(q, a, q') \in T'$*. This is the only transition leaving* q*.*
  - *For all* $b \in \mathtt{Out} \cup \{\delta\}$*, there exists a unique* $q_b \in Q'$*,* $q_b \neq q$*, such that* $(q, b, q_b) \in T'$ *and all the states* q_b *are pairwise different. These are the only transitions leaving* q*.*
  - *The only outgoing transitions are self-loops labelled by each action belonging to* $\mathtt{Out} \cup \{\delta\}$*. In this case we say that the state is* terminal*.*

The language *of* $\mathcal{G}$ *contains all the (finite) traces that can be performed by the scheduler. Overloading the notation used to define the traces of a PIOTS we have* $L(\mathcal{G}) = \{\sigma \in \mathcal{A}ct^* | \exists q' \in Q' : q'_{in} \overset{\sigma}{\Longrightarrow} q'\}$*.*

If a trace σ has been observed then a scheduler can be in a state where it supplies an input or it may be in a state where it waits and observes either output or quiescence (in practical terms, quiescence is usually observed through a timeout). If the scheduler has finished supplying inputs then it may have moved to a leaf state from which it can only observe outputs and quiescence.

At times we will want to be able to define a scheduler that determines the probability, in a system, associated with a particular trace σ. This is achieved using the following type of scheduler, which will prove to be extremely useful in a number of proofs.

Definition 9 *Let* In *and* Out *be sets of inputs and outputs, respectively, and* $\sigma \in \mathcal{A}ct^*$. *The* deterministic scheduler generated by σ, *denoted by* $SG(\sigma)$, *is a deterministic scheduler* $(Q, \texttt{In}, \texttt{Out}, T, q_{in})$ *such that* (Q, T) *are inductively constructed from the initial call* $SG'(\sigma, q_{in}, \{q_{in}\}, \emptyset)$, *as follows:*

$$SG'(\sigma_{aux}, q, Q_{aux}, T_{aux}) = \begin{cases} (Q_{aux}, T_0) & \text{if } \sigma_{aux} = \epsilon \\ SG'(\sigma'_{aux}, q', Q_1, T_1) & \text{if } \sigma_{aux} = x\sigma'_{aux} \wedge x \in \texttt{In} \\ SG'(\sigma'_{aux}, q', Q_2, T_2) & \text{if } \sigma_{aux} = x\sigma'_{aux} \wedge x \in \texttt{Out} \cup \{\delta\} \end{cases}$$

where the state q' *is fresh and* T_0, Q_1, T_1, Q_2, *and* T_2 *are defined as follows:*

$$\begin{array}{l} T_0 = T_{aux} \cup \{(q, a, q) | a \in \texttt{Out} \cup \{\delta\}\} \\ Q_1 = Q_{aux} \cup \{q'\} \\ T_1 = T_{aux} \cup \{(q, x, q')\} \\ Q_2 = Q_1 \cup \{q_a | a \in (\texttt{Out} \cup \{\delta\}) \backslash \{x\}\} \ (\textit{states } q_a \textit{ are fresh}) \\ T_2 = T_1 \cup \{(q, a, q_a), (q_a, y, q_a) | a \in (\texttt{Out} \cup \{\delta\}) \backslash \{x\}, y \in \texttt{Out} \cup \{\delta\}\} \end{array}$$

In Figure 8 (right) we present $SG(?i_1!o'_2?i'_2)$ assuming that $\texttt{In} = \{?i_1, ?i'_2\}$ and $\texttt{Out} = \{!o_1, !o'_2, !o''_2\}$. Let us note that the main difference between general deterministic schedulers and schedulers generated by a trace is that the former can have more than one output branch having a non-trivial continuation (e.g. in the previous graph, some of the leaves in the intermediate level might be further extended). Next we define the application of a deterministic scheduler $\mathcal{G}$ to a system s; this defines a PLTS $s \parallel \mathcal{G}$ that quantifies the probability of observing a trace σ if PIOTS s interacts with scheduler $\mathcal{G}$.

Definition 10 *Let* $s = (Q, \texttt{In}, \texttt{Out}, T, q_{in})$ *be a PIOTS with a set of ports* $\mathcal{O}$ *and* $\mathcal{G} = (Q', \texttt{In}, \texttt{Out}, T', q'_{in})$ *be a deterministic scheduler for* In *and* Out. *We define the application of* $\mathcal{G}$ *to* s, *denoted* $s \parallel \mathcal{G}$, *as the PLTS* $s' = (Q'', \mathcal{A}ct, T'', (q_{in}, q'_{in}))$ *such that* $Q'' \subseteq Q \times Q'$ *is the set of states reachable from the initial state under the set of transitions* T''. *We have that* $((q_1, q'_1), a, (q_2, q'_2), p) \in T''$ *if and only if* $a \in \mathcal{A}ct$, $(q_1, a, q_2, p) \in T$ *and* $(q'_1, a, q'_2) \in T'$.

It is trivial to prove that this composition produces a PLTS.

Example 2 *The application of the deterministic schedulers presented in Figure 8 to the PIOTS presented in Figure 7 produce the PLTSs given in Figure 9.*

Since we consider PIOTSs that are not output-divergent, the composition of a deterministic scheduler and a process defines a PLTS with only finitely many traces that do not end in δ. To see that this is the case, first note that a deterministic scheduler is a *finite* tree whose leaves are labelled with output loops. Thus, the only way in which the composition of a deterministic scheduler and a process could have infinitely many traces that end in a single δ is if this composition can reach a state where the deterministic scheduler is in a leaf state; the composition can then follow an infinite path in which all of the transitions

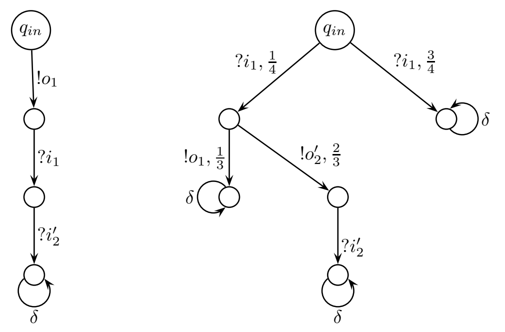

Figure 9: Application of schedulers to PIOTSs producing PLTSs.

are labelled with outputs. However, this is not possible since processes are not output divergent.

Since the composition of a deterministic scheduler and a process defines a PLTS with only finitely many traces that do not end in δ, the implementation relation $\equiv$ is suitable. The next result [17] makes use of this property to show that the application of deterministic schedulers to processes keeps the symmetry of the $\sqsubseteq$ relation.

Proposition 1 *Let $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ and $\mathcal{G}_r, \mathcal{G}_s$ be deterministic schedulers for* In *and* Out*. Then we have that $r \parallel \mathcal{G}_r \sqsubseteq s \parallel \mathcal{G}_s$ if and only if $s \parallel \mathcal{G}_s \sqsubseteq r \parallel \mathcal{G}_r$.*

Two implementation relations have been defined using deterministic schedulers and $\equiv$ [17].[3] First we define the stronger of the two implementation relations, which requires that for any deterministic scheduler $\mathcal{G}$ we have that the PLTSs that result from combining the implementation and specification PIOTSs with $\mathcal{G}$ are equivalent under $\equiv$.

Definition 11 *Given $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, we write $r \equiv_d^s s$ if for all $\mathcal{G}$, deterministic scheduler for* In *and* Out*, we have $r \parallel \mathcal{G} \equiv s \parallel \mathcal{G}$.*

This implementation relation requires the same scheduler to be used when comparing two processes. In effect, it therefore requires that we know how the environment behaved when interacting with the system under test (the environment is modelled as a scheduler). However, it may not be reasonable to assume that we can know how the environment behaves and so, when comparing PIOTSs r and s we need to allow the possibility that we have used different schedulers. This leads to the following weaker implementation relation [17].

[3]The original names of the relations were $\equiv_g^s$ and $\sqsubseteq_g^w$, where the g stands for *global*, as our deterministic schedulers were formerly called.

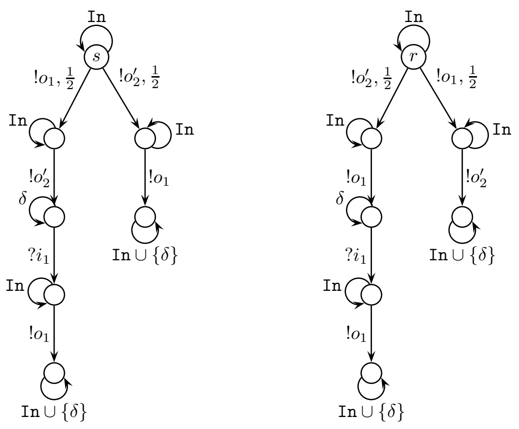

Figure 10: PIOTSs s and r where $r \sqsubseteq_d^w s$ but we do not have that $r \equiv_d^s s$

Definition 12 *Given $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, we write $r \sqsubseteq_d^w s$ if for all $\mathcal{G}_r$, deterministic scheduler for* In *and* Out*, there exists $\mathcal{G}_s$, deterministic scheduler for* In *and* Out*, such that $r \parallel \mathcal{G}_r \equiv s \parallel \mathcal{G}_s$.*

It is obvious that $r \equiv_d^s s$ implies $r \sqsubseteq_d^w s$. We show in Figure 10 that, in general, $r \sqsubseteq_d^w s$ does not imply $r \equiv_d^s s$ [17].

The rest of the paper explores how these two implementation relations change if we use probabilistic schedulers.

5. Extending schedulers with probabilistic choice

The schedulers we defined in the previous section were *deterministic* in the following sense: after observing a trace either they wait to observe output (or quiescence) or they attempt to apply a unique input. An alternative is to allow *probabilistic schedulers*. In this section we introduce the notion of a probabilistic schedulers and define the composition of a probabilistic scheduler and a PIOTS. We then show how one can determine the probability of such a composition producing trace σ; in the next section we will use this to define new implementation relations that use probabilistic schedulers.

There are a number of ways of defining probabilistic schedulers. One option would be to have a scheduler that, after a trace, can be in a state where it can apply any one of a number of inputs (or choose not to apply an input), with probabilities being placed on these options. We have decided to use an alternative, but essentially equivalent, way of introducing probabilities: we chose to include a new (unobservable) action τ to label transitions associated with probabilistic choice. This choice will ease the task of reasoning about the application of schedulers to systems because the *relevant* probabilistic information will be always attached to τ transitions. In addition, since observations are (linear) traces, it is easy to *simulate* the previously mentioned alternative notion, with probabilistic choices between different inputs, with our probabilistic schedulers. Essentially, our probabilistic schedulers will have a number of states in which they can make probabilistic choices and all transitions leaving such a state s will be labelled τ and have an associated probability. The other states will be similar to the states of deterministic schedulers. Thus, a probabilistic scheduler will use probabilistic choice over τ transitions to determine how it should behave. We will let $\mathcal{A}ct^\tau$ denote the extended alphabet $\mathcal{A}ct \cup \{\tau\}$.

Definition 13 *Let* `In` *and* `Out` *be sets of inputs and outputs, respectively. A* probabilistic scheduler *for* `In` *and* `Out` *is a tuple* $\mathcal{G} = (Q', \mathtt{In}, \mathtt{Out}, T', q'_{in})$ *such that* Q' *is a finite set of states, with* $q'_{in} \in Q'$ *its initial state, and* $T' \subseteq Q' \times \mathcal{A}ct^\tau \times Q' \times [0,1]$ *is its transition relation. The transition relation* T' *satisfies the following conditions:*

- *The graph induced by* T' *is a tree with the exception of its* leaves*, which will have self-loop transitions labelled by outputs and* δ*, all with probability* 1.
- *For all* $q \in Q'$*, one of the following possibilities hold:*
  - *There exist probabilities* $p_1, \ldots, p_k \in (0,1]$ *and distinct states* $q_1, \ldots, q_k \in Q \backslash \{q\}$ *such that for all* $1 \leq j \leq k:\ (q, \tau, q_j, p_j) \in T'$ *and* $\sum_j p_j = 1$*. These are the only transitions leaving* q.
  - *There exists one* $a \in \mathtt{In}$ *and* $q' \neq q$ *such that* $(q, a, q', 1) \in T'$*. This is the only transition leaving* q.
  - *For all* $b \in \mathtt{Out} \cup \{\delta\}$*, there exists a unique* $q_b \in Q'$*,* $q_b \neq q$*, such that* $(q, b, q_b, 1) \in T'$ *and all the* q_b *are pairwise different. These are the only transitions leaving* q.
  - *The only outgoing transitions are self-loops labelled by each action belonging to* $\mathtt{Out} \cup \{\delta\}$*, all having probability* 1*. In this case we say that the state is* terminal.

The language *of* $\mathcal{G}$ *contains all the (finite) traces of visible actions that can be performed by the probabilistic scheduler. Formally,* $a_1 \ldots a_n \in \mathcal{A}ct^*$ *belongs to* $L(\mathcal{G})$ *if there exist* $q_1, q'_1, \ldots, q_n, q'_n$ *such that* $q'_{in} \xrightarrow{\tau^*, p_1} q_1$*,* $q_1 \xrightarrow{a_1, 1} q'_1$*,* $q'_1 \xrightarrow{\tau^*, p_2} q_2$*,* $q_2 \xrightarrow{a_2, 1} q'_2$*, …,* $q'_{n-1} \xrightarrow{\tau^*, p_n} q_n$ *and* $q_n \xrightarrow{a_n, 1} q'_n$*, where we have that* $q \xrightarrow{\tau^*, p} q'$ *holds if there is a sequence of zero or more consecutive* τ *transitions from* q *to* q' *such that the product of their associated probabilities is equal to* p *(we assume that* $q \xrightarrow{\tau^*, 1} q$ *holds).*

Deterministic schedulers are clearly special cases of probabilistic schedulers in which there are no τ transitions and so all transitions are labelled with probability 1. The addition of *artificial* probability 1, to transitions labelled with actions in $\mathcal{A}ct$, is a technicality to simplify the computation of probabilities associated with sequences of actions. Specifically, if we apply a probabilistic scheduler to a system, to obtain a PLTS, we will multiply the probabilities of the actions in the system by the probability of the same action in the scheduler. Since the latter will always be equal to 1 we simply obtain the former values.

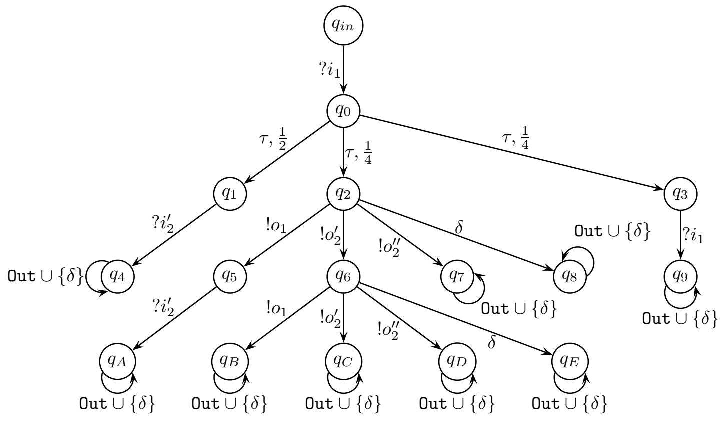

Figure 11: Example of a probabilistic scheduler

Therefore, these *probabilities* will not influence the probabilistic choices taken in the PIOTSs where probabilistic schedulers will be applied. In Figure 11 we give an example of a probabilistic scheduler, where as usual $\texttt{In} = \{?i_1, ?i'_2\}$ and $\texttt{Out} = \{!o_1, !o'_2, !o''_2\}$. We now formally define how a probabilistic scheduler is composed with a PIOTS.

Definition 14 *Let $s = (Q, \texttt{In}, \texttt{Out}, T, q_{in})$ be a PIOTS and $\mathcal{G} = (Q', \texttt{In}, \texttt{Out}, T', q'_{in})$ be a probabilistic scheduler. We define the application of the probabilistic scheduler $\mathcal{G}$ to s, denoted by $s \parallel \mathcal{G}$, as the labelled transition system $(Q'', \mathcal{A}ct^\tau, T'', (q_{in}, q'_{in}))$ such that $Q'' \subseteq Q \times Q'$ is the set of states reachable from the initial state (q_{in}, q'_{in}) under the set of transitions T''. We have that $((q_1, q'_1), a, (q_2, q'_2), p) \in T''$ if and only if one of the following holds.*

1. *$a = \tau$, $q_1 = q_2$, and $(q'_1, \tau, q'_2, p) \in T'$.*
2. *$a \in \mathcal{A}ct$, $(q_1, a, q_2, p_1) \in T$, $(q'_1, a, q'_2, 1) \in T'$, and $p = p_1$.*

In the appendix of the paper (see Proposition 3) we prove that the composition of a PIOTS and a probabilistic scheduler produces a PLTS.

We now need to generalise our previous definition of the probability of a trace σ being observed since this was for processes that do not have τ transitions. First, we define the notion of *path*: a sequence of consecutive transitions, whose labels are from $\mathcal{A}ct^\tau$, that can be performed. In order to avoid the repetition of the definition, we will introduce the concept of label for probabilistic schedulers and PLTSs.

Definition 15 *Let t be either a PLTS or a probabilistic scheduler. A* path *of t is a sequence $(q_{in}, a_1, q_2, p_1)(q_2, a_2, q_3, p_2) \dots (q_k, a_k, q_{k+1}, p_k)$ of consecutive transitions. We denote by $Paths(t)$ the set of paths of t. If ρ is a path then $pre(\rho)$ denotes the set of prefixes of ρ and so $pre((q_{in}, a_1, q_2, p_1) \dots (q_k, a_k, q_{k+1}, p_k)) = \{(q_{in}, a_1, q_2, p_1) \dots (q_i, a_i, q_{i+1}, p_i) | 1 \le i \le k\}$. Given path ρ we let $label(\rho)$ be* *inductively defined as follows.*

$$label(\rho) = \begin{cases} \epsilon & \text{if } \rho = \epsilon \\ a\ label(\rho') & \text{if } \rho = (q, a, q', p)\rho' \ \wedge \ a \neq \tau \\ label(\rho') & \text{if } \rho = (q, \tau, q', p)\rho' \end{cases}$$

Finally, given $\rho \in Paths(t)$ we let $prob(t, \rho)$ be inductively defined as follows.

$$prob(t, \rho) = \begin{cases} 1 & \text{if } \rho = \epsilon \\ p \cdot prob(\rho') & \text{if } \rho = (q, a, q', p)\rho' \end{cases}$$

For completeness, we let $prob(t, \rho) = 0$ if ρ does not belong to $Paths(t)$.

Let us suppose that σ is the label of a path of a process t that contains transitions with label τ. In order to assign the probability of observing σ from t, we cannot simply sum the probabilities of the paths with label σ since, for example, if path ρ has label σ and can be followed by a transition (q, τ, q', p) then path $\rho(q, \tau, q', p)$ also has label σ; if we include the probabilities of both then we essentially 'double count'. We will therefore sum the probabilities of the *minimal* paths of t that have label σ.

Definition 16 *Let t be either a probabilistic scheduler or a PLTS and $\sigma \in L(t)$ be a trace of t. We let $prob(t, \sigma)$ be defined as follows.*

$$prob(t, \sigma) = \sum_{\rho \in Paths(t)} \{\!\mid prob(t, \rho) | label(\rho) = \sigma \wedge (\rho' \in pre(\rho) \setminus \{\rho\} \Rightarrow label(\rho') \neq \sigma) \mid\!\}$$

For completeness, if σ is not a trace of t then we let $prob(t, \sigma) = 0$.

Given a set of sequences of visible actions $c \subseteq \mathcal{A}ct^*$, we let $prob(t, c) = \sum_{\sigma \in c} prob(t, \sigma)$.

Intuitively, $prob(t, \sigma)$ aggregates the probabilities associated with different paths of t *labelled* by a sequence of visible actions σ such that the path does not end with a non-empty sequence of τs (this is imposed by the second condition of the sum because if ρ has a proper prefix with the same label as ρ then we know that ρ finishes with one or more occurrences of τ).

We now explore how probabilistic schedulers can be used to define implementation relations.

6. Implementation relations

In this section we first define a new implementation relation that extends $\equiv^s_d$ with the use of probabilistic schedulers rather than just deterministic schedulers. We then consider weaker implementation relations.

Definition 17 *Let $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$. We write $r \equiv^s_p s$ if for all $\mathcal{G}$, probabilistic scheduler for $\mathtt{In}$ and $\mathtt{Out}$, we have $r \parallel \mathcal{G} \equiv s \parallel \mathcal{G}$.*

This is a very natural notion of correctness: schedulers are models of the environment and so this says that, irrespective of how the environment behaves, the observations one might make with respect to r and s should be the same and should also have the same associated probabilities. One might use such an implementation relation in development since it allows one to replace a specification s by r and know that this will not change the observations that might be made.

The following is an immediate consequence of Proposition 6 (see the appendix of the paper for the proof).

Theorem 1 *Let $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$. We have that $r \equiv_d^s s$ holds if and only if $r \equiv_p^s s$ holds.*

Thus, if we are interested in the strong implementation relation then it is sufficient to reason about deterministic schedulers. We now consider a weaker notion of correctness and find that here the use of probabilistic schedulers does make a difference. The new implementation relation extends $\sqsubseteq_d^w$ with the use of probabilistic schedulers rather than just deterministic schedulers.

Definition 18 *Let $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$. We write $r \sqsubseteq_p^w s$ if for all $\mathcal{G}_r$, probabilistic scheduler for $\mathtt{In}$ and $\mathtt{Out}$, there exists $\mathcal{G}_s$, probabilistic scheduler for $\mathtt{In}$ and $\mathtt{Out}$, such that $r \parallel \mathcal{G}_r \equiv s \parallel \mathcal{G}_s$.*

Recall that $r \equiv_p^s s$ indicates that the processes have the same probability attached to each class of traces after applying the same probabilistic scheduler; $r \sqsubseteq_p^w s$ means that the application of a probabilistic scheduler to r can always be simulated by the application of another (possibly different) one to s. This captures the situation in which we cannot know how the environment behaves; we can only check that the observations made are consistent with the specification within some (possibly different) environment. This notion of correctness corresponds to the power of testing in situations in which one cannot know the global behaviour of the environment since one is not able to control or observe its global traces.

Example 3 *Consider the PIOTSs s and r, with $\mathtt{In}_1 = \{?i_1\}$, $\mathtt{In}_2 = \emptyset$, $\mathtt{Out}_1 = \emptyset$ and $\mathtt{Out}_2 = \{!o_2, !o_2'\}$ depicted in Figure 12 (left). These processes are not related under $\equiv_p^s$; it is sufficient to consider $SG(!o_2?i_1!o_2')$ and note that*

$$\begin{aligned} prob(s \parallel SG(!o_2?i_1!o_2'), [!o_2?i_1!o_2'\delta]) &= 1 \\ &\neq 0 = prob(r \parallel SG(!o_2?i_1!o_2'), [!o_2?i_1!o_2'\delta]) \end{aligned}$$

We have that $s \sqsubseteq_p^w r$ does not hold either. In this case, it is sufficient to consider again $SG(!o_2?i_1!o_2')$ applied to s and observe that there does not exist a probabilistic scheduler $\mathcal{G}_r$ such that $prob(r \parallel \mathcal{G}_r, [!o_2?i_1!o_2'\delta]) = 1$. On the contrary, we have $r \sqsubseteq_p^w s$. Let $\mathcal{G}_r$ be a scheduler for r. We will show that there exists a scheduler $\mathcal{G}_s$ for s such that $r \parallel \mathcal{G}_r \equiv s \parallel \mathcal{G}_s$. There are three types of relevant schedulers:

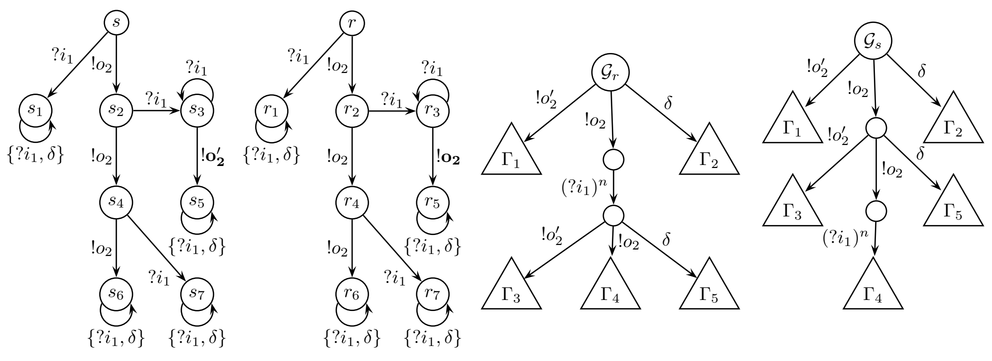

Figure 12: PIOTSs and the relations $\equiv_p^s$ and $\sqsubseteq_p^w$

- *The scheduler initially offers input $?i_1$.*
- *The scheduler initially offers output twice.*
- *The scheduler initially offers output and then offers one or more inputs $?i_1$.*

In the first two cases it is trivial to see that using $\mathcal{G}_r = \mathcal{G}_s$ we have the fulfillment of $\equiv$. In the last case, it is enough to swap all the occurrences of inputs with the first occurrence of $!o_2$ to build the scheduler $\mathcal{G}_s$ (see Figure 12 (right)). This is admissible, in particular, because the inputs and the output are performed at different ports. It can be checked that $r \parallel \mathcal{G}_r \equiv s \parallel \mathcal{G}_s$.

The proof of the following result is an immediate consequence of properties of the PIOTSs presented in the previous example.

Proposition 2 *Let $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$. We have that $r \equiv_p^s s$ implies $r \sqsubseteq_p^w s$ but the converse is not true. In addition, the relation $\sqsubseteq_p^w$ is not symmetric, that is, $r \sqsubseteq_p^w s$ does not imply $s \sqsubseteq_p^w r$.*

Previously, we saw that we do not change the distinguishing power if we consider probabilistic schedulers in the scope of our strong relation. However, this is not the case for the weak versions. The following result is immediate from Propositions 8 and 9 (see the appendix of the paper).

Theorem 2 *Let $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$. We have that if $r \sqsubseteq_d^w s$ holds then $r \sqsubseteq_p^w s$ holds but the converse is not the case.*

Similar to our previous work, which used deterministic schedulers, the weak implementation relation allows the use of different schedulers $\mathcal{G}_r$ and $\mathcal{G}_s$ with r and s. The aim is for this to capture uncertainty regarding how the environment behaves, with this uncertainty being a consequence of the distributed nature of interactions between the SUT and its environment. However, the (human) tester can choose the test case to apply and so has some information regarding how

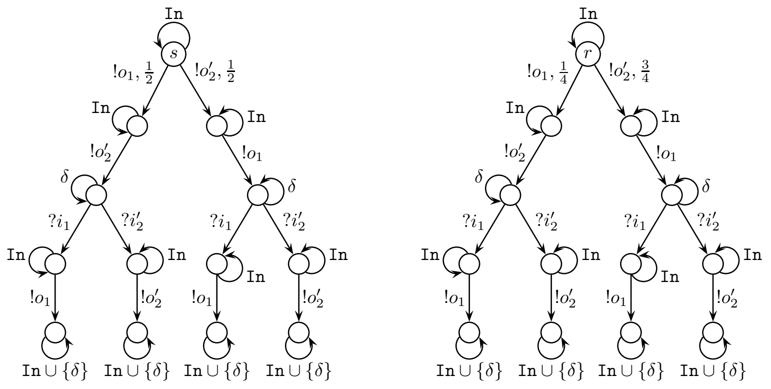

Figure 13: PIOTSs s and r where neither $r \sqsubseteq_d^w s$ nor $s \equiv_p^s r$ hold

the environment behaves during testing, even if they do not have a global view. For example, the tester will have chosen the inputs that can be applied. As a result, one would expect $\mathcal{G}_r$ and $\mathcal{G}_s$ to be related in some way. Specifically, we can prove that the projections of these two schedulers must be the same. In other words, $\mathcal{G}_r$ and $\mathcal{G}_s$ can be different but if we observe them in different ports then the observations coincide (see Proposition 10 in the appendix of the paper).

Finally, we study the relation induced by having both $r \sqsubseteq_p^w s$ and $s \sqsubseteq_p^w r$. We define a new implementation relation, based on this condition, and study how it relates to $\equiv_p^s$.

Definition 19 *Let $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$. We write $r \equiv_p^w s$ if we have both $r \sqsubseteq_p^w s$ and $s \sqsubseteq_p^w r$.*

This relation is trivially reflexive and symmetric. It is easy to see that it is also transitive, because $\sqsubseteq_p^w$ is transitive. Therefore, we have the following result.

Lemma 1 *The relation $\equiv_p^w$ is an equivalence.*

At this point, it is legitimate to ask whether both equivalence relations, $\equiv_p^s$ and $\equiv_p^w$, coincide. However, we have a result similar to the well-known relation between simulation and bisimulation: two processes may simulate each other and still not be bisimilar. We trivially have that $s \equiv_p^s r$ implies $s \equiv_p^w r$. However, the other implication does not hold in general.

Theorem 3 *Let $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$. We have that $s \equiv_p^s r$ implies $s \equiv_p^w r$. However, there exist $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $s \equiv_p^w r$ holds but $s \equiv_p^s r$ does not hold.*

Proof. Consider the PIOTSs s and r shown in Figure 13. Using an argument similar to the one used in the proof of Proposition 9 (see the appendix of the paper) it is easy to see that we have both $r \sqsubseteq_p^w s$ and $s \sqsubseteq_p^w r$. Therefore,

$s \equiv_p^w r$. However, $s \equiv_p^s r$ does not hold. We choose a deterministic scheduler $\mathcal{G}$ that provides input $?i_1$ after $!o_1!o'_2$ but not after $!o'_2!o_1$. This gives $[!o_1!o'_2?i_1!o_1\delta]$ a probability of $\frac{1}{4}$ in $r \parallel \mathcal{G}$ while it gives a probability of $\frac{1}{2}$ in $s \parallel \mathcal{G}$.

7. Conclusions and future work

If the system under test has multiple physically distributed ports, we place a separate independent tester at each port, and there is no global clock then we are testing in the distributed test architecture. Previous work defined implementation relations, for testing from a PIOTS in the distributed test architecture. The initial work noted that an observation defines an equivalence class of traces and one cannot simply sum the probabilities of the traces in such an equivalence class. As a result, the initial work applied only to a restricted class of PIOTS, with this later being generalised through the use of deterministic schedulers to resolve races.

Deterministic schedulers, previously used to define implementation relations, essentially model the environment of the SUT. When testing from probabilistic models it seems natural to allow the SUT's environment to be probabilistic and so in this paper we generalised previous work by defining implementation relations using probabilistic schedulers. Interestingly, it transpired that the introduction of probabilities into schedulers does not change our stronger implementation relation $\equiv_p^s$. However, this stronger implementation relation corresponds to the case where we can know the global behaviour of the environment, and so we can use the same scheduler when comparing the SUT and the specification. In practice, it seems likely that one will not be able to know the global behaviour of the environment and this motivates the definition of a weaker implementation relation. Under this, for r to be a correct implementation of s it is sufficient that for any possible environment (probabilistic scheduler for r), the behaviour of r in this environment is equivalent to the behaviour of s in some (possibly different) environment. This weaker notion of correctness is not an equivalence relation and is weaker than the corresponding implementation relation for deterministic schedulers. One can use the weaker implementation relation $\sqsubseteq_p^w$ to define an equivalence relation $\equiv_p^w$ in which $r \equiv_p^w s$ if and only if we have that $r \sqsubseteq_p^w s$ and $s \sqsubseteq_p^w r$. This implementation relation is an equivalence relation but is strictly weaker than $\equiv_p^s$.

There are several lines of future work. First, we have shown that the pairs of probabilistic schedulers, used under $\sqsubseteq_p^w$, must relate in certain ways but we might wish to place further restrictions on such pairs of probabilistic schedulers. Such restrictions might correspond to information that the tester has about the behaviour of the environment. There is also the question of how one might efficiently generate tests for testing from our new implementation relations.

[1] G. J. Myers, The Art of Software Testing, 2nd Edition, John Wiley and Sons, 2004.

[2] R. M. Hierons, K. Bogdanov, J. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A. Simons,

S. Vilkomir, M. Woodward, H. Zedan, Using formal specifications to support testing, ACM Computing Surveys 41 (2).

[3] M. Utting, A. Pretschner, B. Legeard, A taxonomy of model-based testing approaches, Software Testing, Verification and Reliability 22 (5) (2012) 297–312.

[4] A. R. Cavalli, T. Higashino, M. Núñez, A survey on formal active and passive testing with applications to the cloud, Annales of Telecommunications 70 (3-4) (2015) 85–93.

[5] R. Marinescu, C. Seceleanu, H. L. Guen, P. Pettersson, A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs, Vol. 98 of Advances in Computers, Elsevier, 2015, Ch. 3, pp. 89–140.

[6] M. Shafique, Y. Labiche, A systematic review of state-based test tools, International Journal on Software Tools for Technology Transfer 17 (1) (2015) 59–76.

[7] W. Grieskamp, N. Kicillof, K. Stobie, V. Braberman, Model-based quality assurance of protocol documentation: tools and methodology, Software Testing, Verification and Reliability 21 (1) (2011) 55–71.

[8] I. Hwang, A. R. Cavalli, M. Lallali, D. Verchère, Applying formal methods to PCEP: an industrial case study from modeling to test generation, Software Testing, Verification and Reliability 22 (5) (2012) 343–361.

[9] J. Peleska, Industrial-strength model-based testing - state of the art and current challenges, in: 8th Workshop on Model-Based Testing, MBT'13, EPTCS 111, 2013, pp. 3–28.

[10] C. Braunstein, A. E. Haxthausen, W.-L. Huang, F. Hübner, J. Peleska, U. Schulze, L. V. Hong, Complete model-based equivalence class testing for the ETCS ceiling speed monitor, in: 16th Int. Conf. on Formal Engineering Methods, ICFEM'14, LNCS 8829, Springer, 2014, pp. 380–395.

[11] R. V. Binder, B. Legeard, A. Kramer, Model-based testing: where does it stand?, Communications of the ACM 58 (2) (2015) 52–56.

[12] J. Tretmans, Model based testing with labelled transition systems, in: Formal Methods and Testing, LNCS 4949, Springer, 2008, pp. 1–38.

[13] J. T. C. ISO/IEC JTC 1, International Standard ISO/IEC 9646-1. Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts, ISO/IEC, 1994.

[14] R. M. Hierons, M. G. Merayo, M. Núñez, Implementation relations for the distributed test architecture, in: Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int.

Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047, Springer, 2008, pp. 200–215.

[15] R. M. Hierons, M. G. Merayo, M. Núñez, Implementation relations and test generation for systems with distributed interfaces, Distributed Computing 25 (1) (2012) 35–62.

[16] R. M. Hierons, M. Núñez, Testing probabilistic distributed systems, in: IFIP 30th Int. Conf. on Formal Techniques for Distributed Systems, FMOODS/FORTE'10, LNCS 6117, Springer, 2010, pp. 63–77.

[17] R. M. Hierons, M. Núñez, Using schedulers to test probabilistic distributed systems, Formal Aspects of Computing 24 (4-6) (2012) 679–699.

[18] B. Sarikaya, G. v. Bochmann, Synchronization and specification issues in protocol testing, IEEE Transactions on Communications 32 (1984) 389–395.

[19] R. Dssouli, G. v. Bochmann, Error detection with multiple observers, in: 5th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'85, North-Holland, 1985, pp. 483–494.

[20] R. Dssouli, G. v. Bochmann, Conformance testing with multiple observers, in: 6th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'86, North-Holland, 1986, pp. 217–229.

[21] S. Boyd, H. Ural, The synchronization problem in protocol testing and its complexity, Information Processing Letters 40 (3) (1991) 131–136.

[22] W. Chen, H. Ural, Synchronizable checking sequences based on multiple UIO sequences, IEEE/ACM Transactions on Networking 3 (1995) 152–157.

[23] R. M. Hierons, H. Ural, UIO sequence based checking sequences for distributed test architectures, Information and Software Technology 45 (12) (2003) 793–803.

[24] G. Luo, R. Dssouli, G. v. Bochmann, Generating synchronizable test sequences based on finite state machine with distributed ports, in: 6th IFIP Workshop on Protocol Test Systems, IWPTS'93, North-Holland, 1993, pp. 139–153.

[25] H. Ural, D. Whittier, Distributed testing without encountering controllability and observability problems, Information Processing Letters 88 (3) (2003) 133–141.

[26] W.-J. Wu, W.-H. Chen, C. Tang, Synchronizable test sequence for multiparty protocol conformance testing, Computer Communications 21 (13) (1998) 1177–1183.

[27] K.-C. Tai, Y.-C. Young, Synchronizable test sequences of finite state machines, Computer Networks and ISDN Systems 30 (12) (1998) 1111–1134.

[28] L. Cacciari, O. Rafiq, Controllability and observability in distributed testing, Information and Software Technology 41 (11–12) (1999) 767–780.

[29] O. Rafiq, L. Cacciari, Coordination algorithm for distributed testing, The Journal of Supercomputing 24 (2) (2003) 203–211.

[30] A. Khoumsi, T. Jéron, H. Marchand, Test cases generation for nondeterministic real-time systems, in: 3rd Int. Workshop on Formal Approaches to Testing of Software, FATES'03, LNCS 2931, Springer, 2003, pp. 131–146.

[31] R. M. Hierons, Reaching and distinguishing states of distributed systems, SIAM Journal on Computing 39 (8) (2010) 3480–3500.

[32] R. M. Hierons, H. Ural, The effect of the distributed test architecture on the power of testing, The Computer Journal 51 (4) (2008) 497–510.

[33] S. Haar, C. Jard, G.-V. Jourdan, Testing input/output partial order automata, in: Joint 19th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'07, and 7th Int. Workshop on Formal Approaches to Software Testing, FATES'07, LNCS 4581, Springer, 2007, pp. 171–185.

[34] G. v. Bochmann, S. Haar, C. Jard, G.-V. Jourdan, Testing systems specified as partial order input/output automata, in: Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047, Springer, 2008, pp. 169–183.

[35] H. Ponce de León, S. Haar, D. Longuet, Unfolding-based test selection for concurrent conformance, in: 25th IFIP WG 6.1 Int. Conf. on Testing Software and Systems, ICTSS'13, LNCS 8254, Springer, 2013, pp. 98–113.

[36] H. Ponce de León, S. Haar, D. Longuet, Model-based testing for concurrent systems: unfolding-based test selection, International Journal on Software Tools for Technology Transfer 18 (3) (2016) 305–318.

[37] H. Ponce de León, S. Haar, D. Longuet, Distributed testing of concurrent systems: Vector clocks to the rescue, in: 11th Int. Colloquium on Theoretical Aspects of Computing, ICTAC'14, LNCS 8687, Springer, 2014, pp. 369–387.

[38] I. Christoff, Testing equivalences and fully abstract models for probabilistic processes, in: 1st Int. Conf. on Concurrency Theory, CONCUR'90, LNCS 458, Springer, 1990, pp. 126–140.

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

[40] R. Segala, Testing probabilistic automata, in: 7th Int. Conf. on Concurrency Theory, CONCUR'96, LNCS 1119, Springer, 1996, pp. 299–314.

[41] C. Gregorio, M. Núñez, Denotational semantics for probabilistic refusal testing, in: 1st Int. Workshop on Probabilistic Methods in Verification, PROBMIV'98, ENTCS 22, Elsevier, 1999, pp. 111–137.

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

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

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

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

[46] R. M. Hierons, M. G. Merayo, Mutation testing from probabilistic and stochastic finite state machines, Journal of Systems and Software 82 (11) (2009) 1804–1818.

[47] I. Hwang, A. Cavalli, Testing a probabilistic FSM using interval estimation, Computer Networks 54 (7) (2010) 1108–1125.

[48] Y. Deng, A. Tiu, Characterisations of testing preorders for a finite probabilistic π-calculus, Formal Aspects of Computing 24 (4-6) (2012) 701–726.

[49] S. Georgievska, S. Andova, Probabilistic may/must testing: retaining probabilities by restricted schedulers, Formal Aspects of Computing 24 (4-6) (2012) 727–748.

[50] D. P. Gruska, Information flow testing, Fundamenta Informaticae 128 (1-2) (2013) 81–95.

[51] Y. Deng, R. v. Glabbeek, M. Hennessy, C. Morgan, Real-reward testing for probabilistic processes, Theoretical Computer Science 538 (2014) 16–36.

[52] M. Gerhold, M. Stoelinga, Model-based testing of probabilistic systems, in: 19th Int. Conf. on Fundamental Approaches to Software Engineering, FASE'16, LNCS 9633, Springer, 2016, pp. 251–268.

[53] R. v. Glabbeek, S. Smolka, B. Steffen, Reactive, generative and stratified models of probabilistic processes, Information and Computation 121 (1) (1995) 59–80.

[54] M. Bravetti, A. Aldini, Discrete time generative-reactive probabilistic processes with different advancing speeds, Theoretical Computer Science 290 (1) (2003) 355–406.

[55] R. Segala, A compositional trace-based semantics for probabilistic automata, in: 6th Int. Conf. on Concurrency Theory, CONCUR'95, LNCS 962, Springer, 1995, pp. 234–248.

[56] R. Segala, N. Lynch, Probabilistic simulations for probabilistic processes, Nordic Journal of Computing 2 (2) (1995) 250–273.

[57] R. Segala, Modeling and verification of randomized distributed real-time systems, Ph.D. thesis, MIT, Dept. of Electrical Engineering and Computer Science (1995).

[58] L. Cheung, N. Lynch, R. Segala, F. Vaandrager, Switched PIOA: Parallel composition via distributed scheduling, Theoretical Computer Science 365 (1-2) (2006) 83–108.

Appendix A: Composition of a PIOTS and a probabilistic scheduler is a PLTS

Proposition 3 *Let $s = (Q, \mathtt{In}, \mathtt{Out}, T, q_{in})$ be a PIOTS and $\mathcal{G} = (Q', \mathtt{In}, \mathtt{Out}, T', q'_{in})$ be a probabilistic scheduler. We have that $s \parallel \mathcal{G}$ is a PLTS with alphabet $\mathcal{A}ct^{\tau} = \mathtt{In} \cup \mathtt{Out} \cup \{\delta, \tau\}$.*

Proof. Since we only consider states reachable from the initial state (q_{in}, q'_{in}), it is enough to show that the sum of the probabilities departing from each pair $(q, q') \in Q \times Q'$ is equal to 1. We consider three cases:

1. All the transitions leaving q' are labelled by τ. The first rule of the definition of T'' (see Definition 14) will produce a transition for each of the transitions departing from q'. The addition of the associated probabilities will be 1 because the probabilities are taken from the scheduler.
2. The unique transition departing from q' is an input $?i$. Since s is input-enabled, there are transitions labelled by $?i$ leaving q. The second rule of the definition of T'' will generate a transition for each of these transitions. Again, the addition of the corresponding probabilities is equal to 1 because the probabilities are, in this case, the ones from the PIOTS.
3. There are transitions leaving q' labelled by each action in $\mathtt{Out} \cup \{\delta\}$ and these are all the transitions departing from q'. We distinguish two cases:
   (a) q has outgoing transitions labelled by outputs. In this case, we also know that q cannot have outgoing transitions labelled by δ. The second rule of the definition of T'' will generate a transition for each of these transitions and since the addition of the associated probabilities in s is equal to 1, so it is in the composition.
   (b) q does not have outgoing transitions labelled by outputs. In this case, q must have an outgoing transition labelled by δ (because it is a quiescent state) and since this transition has probability 1, again applying the second rule of the definition of T'', we have the same probability in the composition.

Appendix B: Auxiliary results concerning the relation between $\equiv_p^s$ and $\equiv_d^s$

In this appendix we study results that are used to prove Theorem 1. Specifically, we explore properties of probabilistic schedulers that will be used to reason about $\equiv_p^s$. We will be reasoning about equivalence classes of traces. It will transpire that, given a trace σ of $r \parallel \mathcal{G}$, we will want to reason about certain special prefixes of traces in $[\sigma]$: those that are the longest prefixes that end in inputs.

Definition 20 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of visible actions. We define $pre_{\mathtt{In}}(\sigma)$ to be the longest prefix of σ that ends in input (if σ contains no inputs then we let $pre_{\mathtt{In}}(\sigma) = \epsilon$).*

Let us suppose that σ is a trace of the composition of a scheduler and a process. If we consider a path of the composition $r \parallel \mathcal{G}$ that has label σ then the probability of σ in $\mathcal{G}$ must, by definition, be the same as the probability of $pre_{\mathtt{In}}(\sigma)$ in $\mathcal{G}$. A number of proofs will use this result.

Definition 21 *Let $\sigma_i \in \mathcal{A}ct^*$ be a sequence of visible actions such that there exists $\sigma' \in [\sigma]$ with $\sigma_i = pre_{\mathtt{In}}(\sigma')$. We define $c(\sigma, \sigma_i)$, the set of traces generated by σ_i, to be the set of traces in $[\sigma]$ that have σ_i as a prefix. If σ can be deduced from the context then we simply write c_i. We define $\mathcal{C}_\sigma$ to be the set $\{c_i | \exists \sigma' \in [\sigma] : \sigma_i = pre_{\mathtt{In}}(\sigma')\}$.*

Example 4 *Let $\sigma = ?i_1?i_2'!o_1\delta$ be a sequence of visible actions. If we consider the probabilistic scheduler depicted in Figure 11, then we have two longest prefixes ending in input corresponding to sequences belonging to $[\sigma]$: $\sigma_1 = ?i_1?i_2'$ (sequence, including a τ, from q_{in} to q_4) and $\sigma_2 = ?i_1!o_1?i_2'$ (sequence, including a τ, from q_{in} to q_A). Therefore, we have $c(\sigma, \sigma_1) = \{?i_1?i_2'!o_1\delta\}$ (we will use the shorthand c_1) and $c(\sigma, \sigma_2) = \{?i_1!o_1?i_2'\delta\}$ (we will use the shorthand c_2).*

Proposition 4 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of visible actions. We have that $\mathcal{C}_\sigma$ is a partition of $[\sigma]$.*

Let $\mathcal{G}$ be a probabilistic scheduler, $\sigma \in \mathcal{A}ct^*$ be a trace of $\mathcal{G}$, and $c_i \in \mathcal{C}_\sigma$ be the set of traces generated by σ_i. For all $\sigma' \in c_i$ we have that $prob(\mathcal{G}, \sigma') = prob(\mathcal{G}, \sigma_i)$.

The proof of the first result is easy, by the definition of $\mathcal{C}_\sigma$, while the proof of the second result relies on the fact that a probabilistic scheduler does not attach probabilities other than 1 to outputs and δ. In addition, regardless of the chosen path, once the last input is observed we have that all outputs and δ are always available. Note that $prob(\mathcal{G}, \sigma_i)$ can be equal to zero if there is no trace in $\mathcal{G}$ having prefix σ_i, despite the fact that σ_i is the prefix of a sequence of actions that cannot be distinguished from σ, a trace of $\mathcal{G}$. For example, consider the probabilistic scheduler $\mathcal{G}$ depicted in Figure 11. We have, on the one hand, $prob(\mathcal{G}, ?i_2'?i_1) = 0$ while, on the other hand, $prob(\mathcal{G}, \sigma) = \frac{1}{2}$ for all σ being a sequence of visible actions starting with $?i_1?i_2'$ and followed by one or more actions belonging to $\mathtt{Out} \cup \{\delta\}$. In addition, we have that $?i'_2?i_1$ is the prefix of a sequence of actions belonging to $[\sigma]$ (for instance, if $\sigma = ?i_1?i'_2!o_1$ then $?i'_2?i_1$ is the prefix of $?i'_2?i_11$ and this sequence belongs to $[\sigma]$). A trivial, but useful, corollary of the previous result is the following.

Corollary 1 *Let $\mathcal{G}$ be a probabilistic scheduler and $\sigma \in \mathcal{A}ct^*$ be a trace of $\mathcal{G}$. We have that $prob(\mathcal{G}, [\sigma]) = \sum_{c \in \mathcal{C}_\sigma} prob(\mathcal{G}, c)$.*

We now show how the different probabilities $prob(r \parallel G, c_i)$ relate to the sum of probabilities defined by $SG(\sigma_i)$, the deterministic scheduler generated by σ_i (see Definition 9).

Proposition 5 *Let $s = (Q, \mathtt{In}, \mathtt{Out}, T, q_{in})$ be a PIOTS, $\mathcal{G} = (Q', \mathtt{In}, \mathtt{Out}, T', q'_{in})$ be a probabilistic scheduler, and $\sigma\delta$ be a trace of $s \parallel \mathcal{G}$. For each $c(\sigma, \sigma_i) \in \mathcal{C}_\sigma$ the following holds*

$$prob(s \parallel \mathcal{G}, c(\sigma, \sigma_i)) = prob(\mathcal{G}, \sigma_i) \cdot prob(s \parallel SG(\sigma_i), c(\sigma, \sigma_i))$$

Proof. During the rest of the proof we use c_i to denote $c_i(\sigma, \sigma_i)$. We first observe that, by definition, we have that

$$prob(s \parallel \mathcal{G}, c_i) = \sum_{\sigma' \in c_i} prob(s \parallel \mathcal{G}, \sigma')$$

Let $P = \{\rho \in Paths(\mathcal{G}) | label(\rho) = \sigma_i \wedge (\rho' \in pref(\rho) \setminus \{\rho\} \Rightarrow label(\rho') \neq \sigma_i)\}$ be the set of minimal paths of $\mathcal{G}$ that have label σ_i. We will use proof by induction on the length of the longest path in P. In the base case we have that $P = \{\epsilon\}$ and so $\sigma_i = \epsilon$. But then $c_i = \{\epsilon\}$ and so $prob(\mathcal{G}, \sigma_i) = 1$, $prob(s \parallel \mathcal{G}, c_i) = 1$ and $prob(s \parallel SG(\sigma_i), c_i) = 1$. The result therefore follows.

We now consider the inductive case. The inductive hypothesis is that the result holds if all paths in P have length less than k ($k > 0$) and we suppose that the length of the longest path in P is k. Note that either all paths in P start with a transition with label τ or there is some $a \in \mathcal{A}ct$ such that all paths in P start with a transition with label a. We now consider these two cases.

Case 1: the label of the first transition of all paths in P is τ. Let us suppose that P contains distinct paths $tr_1\rho_1, \ldots, tr_\ell\rho_\ell$ for transitions $tr_1, \ldots, tr_\ell$ and paths $\rho_1, \ldots, \rho_\ell$. Let p_i be the probability associated with tr_i and let $\mathcal{G}_i$ be the probabilistic scheduler defined by starting $\mathcal{G}$ after tr_i (that is, we consider $\mathcal{G}$ but replace its initial state by the ending state of tr_i). By the definition of $s \parallel \mathcal{G}$, we have that

$$prob(s \parallel \mathcal{G}, c_i) = \sum_{j=1}^{\ell} p_j \cdot prob(s \parallel \mathcal{G}_j, c_i)$$

By the inductive hypothesis, we know that

$$prob(s \parallel \mathcal{G}_j, c_i) = prob(\mathcal{G}_j, \sigma_i) \cdot prob(s \parallel SG(\sigma_i), c_i)$$

We now have that

$$prob(s \parallel \mathcal{G}, c_i) = \sum_{j=1}^{\ell} p_j \cdot prob(s \parallel \mathcal{G}_j, c_i) = \sum_{j=1}^{\ell} p_j \cdot prob(\mathcal{G}_j, \sigma_i) \cdot prob(s \parallel SG(\sigma_i), c_i)$$

By construction we also have that $prob(\mathcal{G}, \sigma_i) = \sum_{j=1}^{\ell} p_j \cdot prob(\mathcal{G}_j, \sigma_i)$. We therefore have that $prob(s \parallel \mathcal{G}, c_i) = prob(\mathcal{G}, \sigma_i) \cdot prob(s \parallel SG(\sigma_i), c_i)$ as required.

Case 2: the label of the first transition of each path of P is a for some $a \in \mathcal{A}ct$. Note that, by the definition of a probabilistic scheduler, we therefore have that all paths in P start with the same transition. Let us suppose that P contains distinct paths $tr\rho_1, \ldots, tr\rho_\ell$ for transition tr and paths $\rho_1, \ldots, \rho_\ell$. Define σ' such that $\sigma = a\sigma'$. Further, let $\mathcal{G}'$ be the probabilistic scheduler produced by starting $\mathcal{G}$ in the state reached by tr and let c_i' be the set of traces formed by deleting prefix a from the traces in c_i (that is, $c_i' = \{\gamma | a\gamma \in c_i\}$). By definition, we have that $prob(s \parallel \mathcal{G}, c_i) = \sum \{\!| \; p \cdot prob(\hat{q} \parallel \mathcal{G}', c_i') | (q_{in}, a, q, p) \in T \; |\!\}$, where $\hat{q}$ is equal to s but with q as the initial state (instead of q_{in}). Further, by the inductive hypothesis, for all $(q_{in}, a, q, p) \in T$ we have that $prob(\hat{q} \parallel \mathcal{G}', c_i') = prob(\mathcal{G}', \sigma_i') \cdot prob(\hat{q} \parallel SG(\sigma_i'), c_i')$, where σ_i' is such that $\sigma_i = a\sigma_i'$. We therefore have that

$$prob(s \parallel \mathcal{G}, c_i) = \sum \{\!| \; p \cdot prob(\mathcal{G}', \sigma_i') \cdot prob(\hat{q} \parallel SG(\sigma_i'), c_i') | (q_{in}, a, q, p) \in T \; |\!\}$$

We can pull out the constant term $prob(\mathcal{G}', \sigma_i')$ to give

$$prob(s \parallel \mathcal{G}, c_i) = prob(\mathcal{G}', \sigma_i') \cdot \sum \{\!| \; p \cdot prob(\hat{q} \parallel SG(\sigma_i'), c_i') | (q_{in}, a, q, p) \in T \; |\!\}$$

By the definition of $s \parallel SG(\sigma_i)$, we have

$$prob(s \parallel SG(\sigma_i), c_i) = \sum \{\!| \; p \cdot prob(\hat{q} \parallel SG(\sigma_i'), c_i') | (q_{in}, a, q, p) \in T \; |\!\}$$

and so we get

$$prob(s \parallel \mathcal{G}, c_i) = prob(\mathcal{G}', \sigma_i') \cdot prob(s \parallel SG(\sigma_i), c_i)$$

The result now follows from observing that $prob(\mathcal{G}, \sigma_i) = prob(\mathcal{G}', \sigma_i')$ since the transitions of $\mathcal{G}$ labelled with $a \neq \tau$ all have probability 1.

We now explore how our new implementation relation $\equiv_p^s$, defined in terms of probabilistic schedulers, relates to the corresponding implementation relation $\equiv_d^s$ defined in terms of deterministic schedulers.

Proposition 6 *Let $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$. We have that $r \equiv_d^s s$ holds if and only if for every $\mathcal{G}$, probabilistic scheduler for* In *and* Out*, we have that $r \parallel \mathcal{G} \equiv s \parallel \mathcal{G}$.*

Proof. The right to left implication is immediate from the definitions, since every deterministic scheduler can be seen as a probabilistic scheduler without τ transitions, and so we focus on the left to right implication.

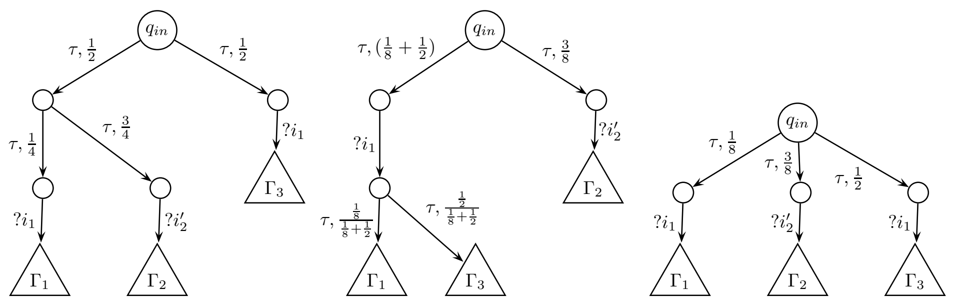

Figure 14: Examples of equivalent probabilistic schedulers

We assume that $r \equiv_d^s s$ and we are given a probabilistic scheduler $\mathcal{G}$. We are required to prove that for all $\sigma \in \mathcal{A}ct^*$ we have that $prob(r \parallel \mathcal{G}, [\sigma\delta]) = prob(s \parallel \mathcal{G}, [\sigma\delta])$.

Let $\sigma_1, \ldots, \sigma_k$ denote the longest prefixes of elements of $[\sigma\delta]$ that end in input. Further, consider $c_1, \ldots, c_k \in \mathcal{C}_{\sigma\delta}$ as given in Definition 21, that is, c_i is the set of traces in $[\sigma\delta]$ that have prefix σ_i. By Proposition 4, the c_i provide a partition of $[\sigma\delta]$ and we have $prob(r \parallel \mathcal{G}, [\sigma\delta]) = \sum_{i=1}^{k} prob(r \parallel \mathcal{G}, c_i)$ and $prob(s \parallel \mathcal{G}, [\sigma\delta]) = \sum_{i=1}^{k} prob(s \parallel \mathcal{G}, c_i)$.

For each σ_i, consider the deterministic scheduler $SG(\sigma_i)$. By Proposition 5, $prob(r \parallel \mathcal{G}, c_i) = prob(\sigma_i, \mathcal{G}) \cdot prob(r \parallel SG(\sigma_i), c_i)$ and $prob(s \parallel \mathcal{G}, c_i) = prob(\sigma_i, \mathcal{G}) \cdot prob(s \parallel SG(\sigma_i), c_i)$. Since $r \equiv_d^s s$, for all $1 \leq i \leq k$ we have that $prob(r \parallel SG(\sigma_i), c_i) = prob(s \parallel SG(\sigma_i), c_i)$. Thus, for all $1 \leq i \leq k$ we have that $prob(r \parallel \mathcal{G}, c_i) = prob(s \parallel \mathcal{G}, c_i)$. The result now follows from observing that $prob(t \parallel \mathcal{G}, [\sigma\delta]) = \sum_{i=1}^{k} prob(t \parallel \mathcal{G}, c_i)$ and $prob(s \parallel \mathcal{G}, [\sigma\delta]) = \sum_{i=1}^{k} prob(s \parallel \mathcal{G}, c_i)$.

Appendix C: Auxiliary results concerning the relation between $\sqsubseteq_d^w$ and $\sqsubseteq_p^w$

In this appendix we study results that are used to prove Theorem 2. First, we introduce an operator to (probabilistically) combine probabilistic schedulers and define a notion of equivalence between probabilistic schedulers.

Definition 22 *Let* `In` *and* `Out` *be sets of inputs and outputs, respectively. Let $\mathcal{G}_1, \ldots, \mathcal{G}_k$ be probabilistic schedulers, with $k \geq 1$ and $\mathcal{G}_j = (Q'_j, \texttt{In}, \texttt{Out}, T'_j, q^j_{in})$, and p_j, with $0 < p_j \leq 1$ be k probabilities such that $\sum p_j = 1$. We define $\sum_{j=1}^{k} p_j \cdot \mathcal{G}_j$ to be the probabilistic scheduler $(\bigcup Q'_j \cup \{q'_{in}\}, \texttt{In}, \texttt{Out}, T', q'_{in})$, where q'_{in} is a fresh state and $T' = \bigcup T'_j \cup \{(q'_{in}, \tau, q^j_{in}, p_j) | 1 \leq j \leq k\}$.*

We say that two probabilistic schedulers $\mathcal{G}$ and $\mathcal{G}'$ are equivalent, denoted by $\mathcal{G} \approx \mathcal{G}'$, if for all PIOTS s we have $s \parallel \mathcal{G} \equiv s \parallel \mathcal{G}'$.

In Figure 14 we give three equivalent probabilistic schedulers. The proof of the next result is trivial. Below we will extend it to show that a probabilistic scheduler is always equivalent to a probabilistic combination of deterministic schedulers.

Lemma 2 *Let s be a PIOTS, $\sigma\delta \in \mathcal{A}ct^*$ be a sequence of visible actions, $\mathcal{G}_1, \ldots, \mathcal{G}_k$ be probabilistic schedulers and consider $\mathcal{G} = \sum_{i=1}^{k} p_i \cdot \mathcal{G}_i$. The following holds*

$$prob(s \parallel \mathcal{G}, [\sigma\delta]) = \sum_{i=1}^{k} p_i \cdot prob(s \parallel \mathcal{G}_i, [\sigma\delta])$$

We now show that a probabilistic scheduler $\mathcal{G}$ is equivalent to a probabilistic choice $\sum_{i=1}^{k} p_i \cdot \mathcal{G}_i$ in which each $\mathcal{G}_i$ is a deterministic scheduler. Essentially, we *lift* all the τs of the probabilistic scheduler and place them as transitions outgoing from the initial state of the resulting probabilistic automata.

Proposition 7 *Let $\mathcal{G}$ be a probabilistic scheduler. There exist deterministic schedulers $\mathcal{G}_1, \ldots, \mathcal{G}_k$ and probabilities $0 < p_1, \ldots, p_k \leq 1$ with $\sum_{i=1}^{k} p_i = 1$ such that $\mathcal{G} \approx \sum_{i=1}^{k} p_i \cdot \mathcal{G}_i$.*

Proof. We will use proof by induction on the length of the longest path in $\mathcal{G}$ (that does not include cycles). The result follows immediately for the base case where all paths are of length 0 since $\mathcal{G}$ must be the trivial scheduler, in which we have a unique state, the initial state, and all the transitions are self-loops, one per each action belonging to $\texttt{Out} \cup \{\delta\}$. This is a deterministic scheduler and it is enough to let $p_1 = 1$.

We now consider the inductive case and use the inductive hypothesis that the result holds for every $\mathcal{G}$ whose longest path has length at most ℓ. There are three cases, which depend on the transitions leaving the initial state of $\mathcal{G}$.

Case 1: The only transition leaving the initial state of $\mathcal{G}$ has label $?i \in \texttt{In}$ and this takes $\mathcal{G}$ to a state q that defines a scheduler $\mathcal{G}'$. By the inductive hypothesis, $\mathcal{G}' \approx \sum_{i=1}^{k} p_i \cdot \mathcal{G}'_i$, for some positive probabilities $p_1, \ldots, p_k$ and deterministic schedulers $\mathcal{G}'_1, \ldots, \mathcal{G}'_k$. The result now follows from the fact that $\mathcal{G} \approx \sum_{i=1}^{k} p_i \cdot \mathcal{G}_i$ in which $\mathcal{G}_i$ applies input $?i$ and then becomes $\mathcal{G}'_i$.

Case 2: All transitions leaving the initial state have label τ. Let us suppose that there are m such transitions and that for all $1 \leq i \leq m$ there is a τ transition with probability p^i to a probabilistic scheduler $\mathcal{G}^i$. By the inductive hypothesis, for all $1 \leq i \leq m$ we have $\mathcal{G}^i \approx \sum_{j=1}^{k_i} p^i_j \cdot \mathcal{G}^i_j$, for some positive probabilities $p^i_1, \ldots, p^i_{k_i}$ and deterministic schedulers $\mathcal{G}^i_1, \ldots, \mathcal{G}^i_{k_i}$. Thus, $\mathcal{G}$ is equivalent to a probabilistic scheduler in which for all $1 \leq i \leq m$ and $1 \leq j \leq k_i$ there is a path with two τ transitions, with probabilities p^i and p^i_j, to a deterministic scheduler $\mathcal{G}^i_j$. This probabilistic scheduler can be rewritten by replacing such a path by a single τ transition with probability $p^i \cdot p^i_j$ to $\mathcal{G}^i_j$. The result thus follows.

Case 3: All transitions leaving the initial state of $\mathcal{G}$ are labelled with an output or δ. Let $\mathcal{G}_a$ denote the probabilistic scheduler reached from the initial state of $\mathcal{G}$ by $a \in \texttt{Out} \cup \{\delta\}$. By the inductive hypothesis, for all $a \in \texttt{Out} \cup \{\delta\}$ we have $\mathcal{G}_a \approx \sum_{j=1}^{k_a} p_j^a \mathcal{G}_j^a$, for some positive probabilities $p_1^a, \ldots, p_{k_a}^a$ and deterministic schedulers $\mathcal{G}_1^a, \ldots, \mathcal{G}_{k_a}^a$. We now apply the following process. For each $a \in \texttt{Out} \cup \{\delta\}$ we choose some $1 \leq i_a \leq k_a$ and this defines a tuple of $\mathcal{G}_{i_a}^a$ with associated probabilities. Let p_1 denote the smallest such probability ($p_1 = \min\{p_{i_a}^a | a \in \texttt{Out} \cup \{\delta\}\}$). Then we construct a deterministic scheduler $\mathcal{G}_1$ that, after output a, becomes the deterministic scheduler $\mathcal{G}_{i_a}^a$. We also give $\mathcal{G}_1$ probability p_1. Further, we reduce the probability associated with each $\mathcal{G}_{i_a}^a$ by p_1, eliminating the $\mathcal{G}_{i_a}^a$ that now have zero probability. This step reduces the number of $\mathcal{G}_j^a$ being considered. We repeat this process until all $\mathcal{G}_j^a$ have been removed. This provides us with a sequence $\mathcal{G}_1, \ldots, \mathcal{G}_k$ of deterministic schedulers and probabilities $p_1, \ldots, p_k$ such that $\mathcal{G} \approx \sum_{i=1}^{k} p_i \cdot \mathcal{G}_i$ and so the result follows.

Example 5 *Consider the equivalent probabilistic schedulers depicted in Figure 14 and suppose that Γ_1, Γ_2 and Γ_3 do not have occurrences of τ. The third probabilistic scheduler has the expected combination of probabilities and deterministic schedulers corresponding to the other two probabilistic schedulers.*

The following shows that the weaker version of our implementation relation for probabilistic schedulers is not stronger than the weaker version for deterministic schedulers.

Proposition 8 *Let $s, r \in \mathcal{PIOTS}(\texttt{In}, \texttt{Out})$. If $r \sqsubseteq_d^w s$ then for every probabilistic scheduler $\mathcal{G}$ there is a probabilistic scheduler $\mathcal{G}'$ such that $r \parallel \mathcal{G} \equiv s \parallel \mathcal{G}'$.*

Proof. We assume that $r \sqsubseteq_d^w s$ and we are given probabilistic scheduler $\mathcal{G}$. We are required to prove that there is a probabilistic scheduler $\mathcal{G}'$ such that for all σ we have that $prob(r \parallel \mathcal{G}, [\sigma\delta]) = prob(s \parallel \mathcal{G}', [\sigma\delta])$.

By Proposition 7 we have that $\mathcal{G}$ is equivalent to a sum $\sum_{j=1}^{k} p_j \cdot \mathcal{G}_j$ of deterministic schedulers. Since $r \sqsubseteq_d^w s$, for all $1 \leq j \leq k$ we have that there exists $\mathcal{G}_j'$ such that for all σ we have that $prob(r \parallel \mathcal{G}_j, [\sigma\delta]) = prob(s \parallel \mathcal{G}_j', [\sigma\delta])$. Thus, we let $\mathcal{G}' = \sum_{j=1}^{k} p_j \cdot \mathcal{G}_j'$ and the result follows.

Proposition 9 *There are PIOTSs r, s such that for every probabilistic scheduler $\mathcal{G}_r$ there is a probabilistic scheduler $\mathcal{G}_s$ such that $r \parallel \mathcal{G}_r \equiv s \parallel \mathcal{G}_s$ but where $r \sqsubseteq_d^w s$ does not hold.*

Proof. Consider the PIOTSs s and r shown in Figure 13 (left and right, respectively). To see that $r \sqsubseteq_d^w s$ does not hold it is sufficient to choose a deterministic scheduler $\mathcal{G}_r$ that provides input $?i_1$ after $!o_1!o_2'$ but not after $!o_2'!o_1$. This gives $[!o_1!o_2'?i_1!o_1\delta]$ a probability of 0.25 in $r||\mathcal{G}_r$ and clearly there is no deterministic scheduler $\mathcal{G}_s$ that provides the same probability for $[!o_1!o_2'?i_1!o_1\delta]$ in $s||\mathcal{G}_s$. Thus, it is sufficient to prove that for every probabilistic scheduler $\mathcal{G}_r$ there is a probabilistic scheduler $\mathcal{G}_s$ such that $r \parallel \mathcal{G}_r \equiv s \parallel \mathcal{G}_s$. Any useful probabilistic scheduler $\mathcal{G}_r$ has choices after $!o_1!o_2'$ and $!o_2'!o_1$. Let p_1 denote its probability of sending $?i_1$ after $!o_1!o'_2$, p_2 denote its probability of sending $?i'_2$ after $!o_1!o'_2$, and p_3 denote its probability of not sending input after $!o_1!o'_2$. Similarly, let p_4 denote its probability of sending $?i'_2$ after $!o'_2!o_1$, p_5 denote its probability of sending $?i'_2$ after $!o'_2!o_1$, and p_6 denote its probability of not sending input after $!o'_2!o_1$. Now we define a probabilistic scheduler $\mathcal{G}_s$ in the following way: after either $!o_1!o'_2$ or $!o'_2!o_1$ the scheduler sends $?i_1$ with probability $\frac{p_1+p_4}{2}$, it sends $?i'_2$ with probability $\frac{p_2+p_5}{2}$, and it sends no input with probability $\frac{p_3+p_6}{2}$. Now observe that there are only four[4] equivalence classes of traces we have to consider under $\equiv$, $[!o_1!o'_2\delta]$, $[!o_1!o'_2?i_1!o_1\delta]$, $[!o_1!o'_2?i'_2!o'_2\delta]$, and $[!o_1!o'_2\delta]$, and the probabilities in both $r||\mathcal{G}_r$ and $s||\mathcal{G}_s$ are 1, $\frac{p_1+p_4}{2}$, $\frac{p_2+p_5}{2}$, and $\frac{p_3+p_6}{2}$ respectively.

We have that Theorem 2 is an immediate consequence of the previous two results.

Finally, we explore the relationship between probabilistic schedulers implied by the definition of $\sqsubseteq_p^w$ and, as a consequence, also how such schedulers must be related for the deterministic case. We start by describing a restriction on the probabilistic schedulers that we need to use. Intuitively, we can restrict ourselves to probabilistic schedulers not having *redundant* nodes, that is, a node at which input is applied such that the interaction of the probabilistic scheduler and the process cannot lead to this node being reached. If a probabilistic scheduler has such a node, for a given system s, then it does not matter the possible continuations after that node because they will never be applied.

Definition 23 *Let $s = (Q, \mathtt{In}, \mathtt{Out}, T, q_{in})$ be a PIOTS and $\mathcal{G} = (Q', \mathtt{In}, \mathtt{Out}, T', q'_{in})$ be a probabilistic scheduler. We say that $\mathcal{G}$ is* non-redundant *for s if and only if, for every state $q' \in Q'$ of $\mathcal{G}$ such that there exists $q'' \in Q'$ and $?i \in \mathtt{In}$ such that $(q', ?i, q'', 1) \in T'$, if σ is the label of the path from q'_{in} to q' then $\sigma \in L(s)$. If this condition does not hold then $\mathcal{G}$ is* redundant *for s.*

Let us suppose that $\mathcal{G}$ has a state q' at which input can be applied such that σ is the label of the path from the root of $\mathcal{G}$ to q' and σ is not a trace of s. Then one can make q' a leaf node and the resultant probabilistic scheduler $\mathcal{G}'$ will be such that $s \parallel \mathcal{G} \equiv s \parallel \mathcal{G}'$. Thus, it is sufficient to consider probabilistic schedulers that are non-redundant for their corresponding processes.

Proposition 10 *Let $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ be PIOTSs and $\mathcal{G}_r, \mathcal{G}_s$ be non-redundant probabilistic schedulers for r and s, respectively. If $r \parallel \mathcal{G}_r \equiv s \parallel \mathcal{G}_s$, then for all $o \in \mathcal{O}$ we have $\pi_o(L(\mathcal{G}_r)) = \pi_o(L(\mathcal{G}_s))$.*

Proof. It is sufficient to prove that $\pi_o(L(\mathcal{G}_r)) \subseteq \pi_o(L(\mathcal{G}_s))$; the opposite direction $\pi_o(L(\mathcal{G}_s)) \subseteq \pi_o(L(\mathcal{G}_r))$ then follows by symmetry. Moreover, by the definition

[4]Formally speaking, we have to consider also those classes with occurrences of inputs and δ in the middle of the trace (for example, the classes $[!o_1!o'_2\delta?i'_2!o'_2\delta]$ and $[!o_1?i_1?i'_2!o'_2?\delta]$). However, in this particular case, the probabilities associated with these classes coincide with the ones corresponding to the classes constructed with traces without these additional occurrences of δ.

of probabilistic schedulers, it is sufficient to consider elements of $\pi_o(L(\mathcal{G}_r))$ that end in input.

Let $\sigma_o \in \pi_o(L(\mathcal{G}_r))$ that ends in input. So there exists some trace $\sigma\delta \in L(\mathcal{G}_r)$ such that σ_o is a prefix of $\pi_o(\sigma)$. Let us suppose that $\sigma = \sigma_1\sigma_2$ such that σ_2 is a longest suffix of σ that contains no inputs. Since $\mathcal{G}_r$ is non-redundant for r we have that $\sigma_1 \in L(r)$ and so $\sigma_1 \in L(r \parallel \mathcal{G}_r)$. Thus, since r is not output-divergent, there is some σ_3 such that $\sigma_1\sigma_3\delta \in L(r \parallel \mathcal{G}_r)$. Since $r \parallel \mathcal{G}_r \equiv s \parallel \mathcal{G}_s$, there is some $\sigma' \sim \sigma_1\sigma_3$ such that $\sigma'\delta \in L(s \parallel \mathcal{G}_s)$. But this implies that $\pi_o(\sigma') \in \pi_o(L(\mathcal{G}_s))$. Since σ_o is a prefix of $\pi_o(\sigma_1\sigma_3)$ and $\sigma' \sim \sigma_1\sigma_3$, we have that σ_o is a prefix of $\pi_o(\sigma')$. But $\pi_o(\sigma') \in \pi_o(L(\mathcal{G}_s))$ and so σ_o is a prefix of a trace of $\pi_o(L(\mathcal{G}_s))$. The result now follows from $\pi_o(L(\mathcal{G}_s))$ being prefix closed.