Under consideration for publication in Formal Aspects of Computing

Using schedulers to test probabilistic distributed systems

Robert M. Hierons[1] and Manuel Núñez[2]

[1]Department of Information Systems and Computing, Brunel University
Uxbridge, Middlesex, UB8 3PH United Kingdom
rob.hierons@brunel.ac.uk
[2]Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, Spain
mn@sip.ucm.es

Abstract. Formal methods are one of the most important approaches to increasing the confidence in the correctness of software systems. A formal specification can be used as an *oracle* in testing since one can determine whether an observed behaviour is allowed by the specification. This is an important feature of formal testing: behaviours of the system observed in testing are compared with the specification and ideally this comparison is automated. In this paper we study a formal testing framework to deal with systems that interact with their environment at physically distributed interfaces, called ports, and where choices between different possibilities are probabilistically quantified. Building on previous work, we introduce two families of schedulers to resolve nondeterministic choices among different actions of the system. The first type of schedulers, which we call *global schedulers*, resolves nondeterministic choices by representing the environment as a single global scheduler. The second type, which we call *localised schedulers*, models the environment as a set of schedulers with there being one scheduler for each port. We formally define the application of schedulers to systems and provide and study different implementation relations in this setting.

Keywords: Distributed systems; formal testing; probabilistic systems; schedulers.

1. Introduction

Carroll Morgan's early work provided significant insights into the relationships between specifications and programs [Mor88, Mor90]. In the mid 1990s he became interested in the study of the effect of probabilities in programming and specification languages. Therefore, he worked on the development of frameworks where

Correspondence and offprint requests to: Robert M. Hierons, Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH United Kingdom, e-mail: rob.hierons@brunel.ac.uk

This research was partially supported by the Spanish MEC project TESIS (TIN2009-14312-C02-01) and the UK EPSRC project Testing of Probabilistic and Stochastic Systems (EP/G032572/1). Part of this research was carried out while the first author was visiting Universidad Complutense de Madrid supported by the *Programa de visitantes distinguidos* UCM - 2011.

probabilities play a fundamental role [MMS96, MMSS96]. Carroll Morgan continued working on probabilistic extensions and, together with colleagues from three continents, this line crystallised into the definition of a new formal testing semantics for probabilistic systems [DGH$^+$07, DGHM08, DGHM09, DGHM11]. Despite there being a myriad of papers on probabilistic testing [Chr90, LS91, YL92, Seg96, CDSY99, Núñ03, LNR06, CSV07, HM09], this proposal was able to find a niche with Carroll Morgan and colleagues presenting a testing theory for a probabilistic extension of CSP. In addition to the usual choice operators, internal and external, the calculus includes a probabilistic (internal) choice. The initial proposal did not include a recursion operator but a later paper [DGHM09] added it to the framework, and then infinite behaviours (in particular, divergent ones) have to be explicitly considered. The semantic theory is a natural generalisation of the classical may and must preorders. In addition to defining these preorders, the work includes alternative characterisations both in terms of simulation relations and in terms of a modal logic. His work has therefore made a significant contribution to specification languages and probabilistic models.

Testing [Mye04, AO08] is the main tool used in industrial environments to increase the confidence in the correctness of software systems. Traditionally, testing has been a *manual* activity and it was usually believed, with a few exceptions [Gau95], that to *formalise* the testing process was a hopeless task. However, and possibly due to the influence of formal approaches to testing of hardware, in recent years there has been growing interest in the area of Model Based Testing (MBT) [HBH08, HBB$^+$09]. In MBT the system under test (SUT) is tested on the basis of a model or specification: the model/specification is used to drive both test generation and test execution. In addition, when testing against a formal specification it is necessary to have an appropriate *implementation relation*, which states what observations regarding the SUT are allowed by the specification. The *standard* relation when testing from an input output transition systems is **ioco** [Tre08], but there are alternatives based on *simulation* relations [LV95]. Typically, work in MBT considers models expressed as either finite state machines (FSMs) or labelled transition systems (LTSs). While developers and testers might not find FSMs and LTSs to be particularly expressive, MBT tools typically take a model or specification written in another language and map this to an FSM or LTS for test generation and execution [GGSV02, Tre08]. MBT thus provides a connection between formal specification languages and formal testing theory and the practical process of testing a piece of software. Since MBT is often automated, it can also provide a strong business case for using formal descriptions. For example, MBT was found to be significantly more cost effective than manual testing in a recent industrial study involving hundreds of testers [GKSB11].

In this paper we consider the testing of systems that interact with their environment at a number of physically distributed interfaces and where the *observations* are thus distributed. In order to test these systems it is necessary to place a separate tester at each port. Therefore, each single tester observes a local trace, which is a projection of the global trace that occurred. For such systems, the observation made during testing is thus a set of local traces rather than a single (global) trace. While this situation has been studied for some years for testing from FSMs or LTSs [SB84, DB85, DB86, BU91, LDB93, CR99, UW03, RC03, HU08], only recently has testing from probabilistic models been considered with our earlier work [HN10] providing a formal testing framework to deal with this type of systems. This first approach considered only a restricted class of probabilistic distributed systems because nondeterminism and probabilities are often difficult to combine. In order to solve this problem, schedulers can be used to resolve nondeterministic choices, so that the resulting systems are fully probabilistic. The development of such an approach is the main goal of this paper. The work complements that of Carroll Morgan and others, who developed specification and modelling languages that have probabilistic choice and associated results, by developing a theory that shows how a system can be tested to check that it conforms to a given specification or model.

We are concerned with systems that interact with their environment at physically distributed ports. We are thus interested in the nature of *distributed observations* and not the structure of the SUT. The SUT may well be implemented as a distributed system but we consider it to be a black-box for the purposes of testing, with this being a normal scenario in system testing[1]. Recent work has looked at the use of schedulers for distributed systems [GD09]. However, this work was concerned with the global behaviour of a system that has separate components and thus investigates different issues: they were interested in the global behaviour of a model comprised of a set of components while we are interested in the distributed external observations that might be made regarding a system.

[1] Most test techniques are either white-box, where they consider the structure of the code, or black-box. Typically, white-box test techniques are only used for the testing of individual components.

Many systems interact with their environment at physically distributed ports. Examples of such systems include communications protocols, web-services, cloud systems and wireless sensor networks. Users perceive these systems as black-boxes and user requirements are thus expressed at this level: users are not interested in the internal structure of a system, only in whether it delivers the services they require. Probabilistic behaviour can come from several sources. First, communications may be unreliable: the (internal or external) delivery of a message may not be ensured. Second, the actual service provided by a system may depend on its current demands, with a busy system providing poorer service. These demands may be external, relating to demands placed on it by other users. The demands may also be internal, since there could be occasional internal activities such as backing up or restructuring data. Finally, some systems are required to be probabilistic in nature, an example being the parts of a communications protocol such as Ethernet that deal with collisions[2].

Next we briefly explain our previous work, give more details about the problems that we confront in this paper and mention some related work. Research on testing systems with physically distributed interfaces has only recently considered models with probabilities [HN10], where two approaches to adding probabilities were presented. First, we considered testing from labelled transition systems where there is a distinction between actions performed at different locations. In this case, we applied a fully *generative* approach [GSS95]. Then we considered systems where in addition to explicitly considering the location where actions were performed we distinguished between inputs and outputs. In this framework a generative approach is not appropriate and we used a combination of the *reactive* [LS91] and generative approaches. Our model is reactive for inputs: given state s and input $?i$, the sum of 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 there are other approaches that are reactive for inputs and generative for outputs [WSS97] or, even without explicitly distinguishing between inputs and outputs, allowing both reactive and generative probabilistic choices between actions [AB00, BA03]. In the latter case, the synchronisation in the context of the parallel operator must involve one reactive occurrence and one generative occurrence of the same action. The interested reader is referred to the original work [GSS95] for longer explanations on the appropriate use of the reactive and generative models and to previous work [BA03] where the usefulness of a mixed reactive-generative model is motivated.

The implementation relations presented in our previous work [HN10] are conservative extensions of previous notions for the non-distributed and/or non-probabilistic framework. For example, if we have only one port and we forget probabilistic information then our implementation relations are equivalent to trace inclusion; if we consider empty sets of inputs then we obtain a natural (probabilistic) extension of trace inclusion in a generative approach. The main problem that we encountered was to determine the probability of making particular observations. Interestingly, it transpires that this can be problematic when we distinguish between inputs and outputs as a result of races. Specifically, observations are not global traces of the system but equivalence classes of global traces that are indistinguishable when there are independent agents/testers at the ports. There can be races between events at different ports and where one or more of these events are inputs the reactive-generative setting does not provide probabilistic information regarding the outcome of such races. As a result, we outlawed these types of races and provided a condition under which such races cannot occur.

In this paper we extend our previous work to solve the aforementioned problem by considering *schedulers*, also called *adversaries* in the literature of probabilistic systems. Schedulers are used to quantify the nondeterminism appearing in systems by modelling possible environments and this overcomes problems caused by races since these races are resolved before probabilities are quantified. Schedulers have been used for a similar purpose in systems combining probabilities and nondeterminism but, to the best of our knowledge, their use in a testing framework of probabilistic systems with distributed interfaces is new. Our implementation relations will depend on the traces that can be observed at different ports. Therefore, our methodology has some similarities with work on semantic notions, in particular testing, for probabilistic automata [Seg95, SL95, Seg96]. The connection with this work can also be established at the modelling level: our mixed reactive-generative interpretation of probabilities can be somehow simulated with the combination of a reactive interpretation of probabilities and the addition of *mixed choices*, that is, a choice between a visible action and a τ invisible one, to relate reactive and generative choices. However, the assumption of distributed ports, the distinction between inputs and outputs and the use of schedulers is not considered in that work. Some of the ideas ap-

[2] In Ethernet, if two nodes have sent messages and these have collided then each node waits a random amount of time before resending its message. Naturally, it would not be appropriate for the nodes to use a fixed amount of time.

pearing in this paper are similar to those found in previous work [CLSV06] 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). Concerning the purpose of schedulers, their use extends our previous work by allowing an additional degree of nondeterminism in which we do not have to forbid races between events at different ports.

The rest of the paper is structured as follows. Section 2 gives preliminary definitions regarding observations that can be made in distributed testing. Section 3 presents a basic implementation relation for fully probabilistic systems that will be used throughout the paper. Section 4 reviews our previous work on testing probabilistic distributed processes with a distinction between inputs and outputs and gives an implementation relation between restricted systems, that is, systems where *pathological* races are forbidden. Section 5 presents our first notion of schedulers. A global scheduler represents a single global agent that provides the environment for the SUT to resolve possibly conflicting situations. We define the application of a global scheduler to a process and consider two scenarios to introduce implementation relations: requiring that the composition of a scheduler and the specification be equivalent to the application of the same scheduler to the SUT (what we call a *strong* relation) and allowing the SUT to choose a different scheduler to simulate the composition of the specification and the original scheduler. Section 6 considers a *local* notion of schedulers: the environment is represented by a set of schedulers with there being one scheduler for each port of the SUT. We define new implementation relations, study their properties, and relate them to the previously defined relations. Finally, Section 7 draws conclusions and discusses future work.

2. Preliminaries

Throughout this paper we assume that there are m observation ports and we identify these using the integers in $\mathcal{O} = \{1, \ldots, m\}$. If $\mathcal{A}ct$ denotes the set of actions then for all $o \in \mathcal{O}$, $\mathcal{A}ct_o$ denotes the set of actions that can be observed at o. As usual, we assume that it is possible to observe the system being in a stable (quiescent) state, this observation being denoted δ. We will include δ in $\mathcal{A}ct$. Quiescence can be observed at all ports and so $\delta \in \mathcal{A}ct_o$ for all $o \in \mathcal{O}$. In the next two sections we will elaborate on the use of quiescence in our formalisms but the interested reader is referred to previous work where quiescence is analysed in a formal testing context [Seg97].

When a system interacts with its environment it does so through a sequence of actions in $\mathcal{A}ct$ called a *global trace*. We denote by $\mathcal{A}ct^*$ the set of global traces and by $\mathcal{A}ct^n$, with $n \in \mathbb{N}$, the set of global traces with length equal to n. Given a global trace $\sigma \in \mathcal{A}ct^*$ we can define the projection $\pi_o(\sigma)$ of σ onto port o, and this is called a *local trace*, in the following way (ϵ represents the empty sequence):

1. $\pi_o(\epsilon) = \epsilon$.
2. If $z \in \mathcal{A}ct_o$ then $\pi_o(z\sigma) = z\pi_o(\sigma)$.
3. If $z \notin \mathcal{A}ct_o$ then $\pi_o(z\sigma) = \pi_o(\sigma)$.

Let us consider, for example, a global trace $a_1b_2c_1$ in which a_1 and c_1 are at port 1 and b_2 is at port 2. Then $\pi_1(a_1b_2c_1) = a_1c_1$ and $\pi_2(a_1b_2c_1) = b_2$.

As stated above, we assume that δ is the only action that can be observed at more than one port. We assume that the $\mathcal{A}ct_o \setminus \{\delta\}$ are pairwise disjoint, adding labels to events if necessary. Given global traces $\sigma, \sigma' \in \mathcal{A}ct^*$ we write $\sigma \sim \sigma'$ if σ and σ' cannot be distinguished when only observing the local traces, that is, $\sigma \sim \sigma'$ if for all $o \in \mathcal{O}$ we have that $\pi_o(\sigma) = \pi_o(\sigma')$. For example, $a_1b_2c_1 \sim b_2a_1c_1$ since $\pi_1(a_1b_2c_1) = a_1c_1 = \pi_1(b_2a_1c_1)$ and $\pi_2(a_1b_2c_1) = b_2 = \pi_2(b_2a_1c_1)$.

The relation $\sim$ is an equivalence relation. Given global trace σ we let $[\sigma]$ denote the equivalence class of σ with respect to $\sim$: the set of global traces indistinguishable from σ when only observing local traces. Thus,

$$[\sigma] = \{\sigma' \in \mathcal{A}ct^* | \forall o \in \mathcal{O} : \pi_o(\sigma') = \pi_o(\sigma)\}$$

In this paper the set $(0, 1]$ denotes all non-zero probabilities; all real numbers that are greater than 0 and no larger than 1. In addition, $[0, 1] = \{0\} \cup (0, 1]$. In general, we use multisets of probabilities, instead of sets, since the same probability can be associated with different transitions whose probabilities we are considering. We use $\{\!|$ and $|\!\}$ as the delimiters for multisets.

The next two tables summarise the main concepts 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. As we said before, an implementation relation defines when an SUT is correct with respect to a specification. At the end of the paper, in Figure 7, we compare the implementation relations described in the paper.

Types of systems		
Notation	*Definition*	*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		
$\sim$	Sec. 2	Relation between traces: two traces are related if all their local projections are equal.
$[\sigma]$	Sec. 2	Contains all the traces that are equivalent to σ with respect to $\sim$.
prob	Def. 2, 3 and 9	This function is overloaded. Given a sequence of actions, it computes the probability of performing the sequence from a state of a PLTS (Definition 2), or all the traces belonging to the equivalence class of the sequence from a state of a PLTS or a PIOTS (Definitions 3 and 9, respectively).
consistent PIOTS	Def. 8	PIOTS not having races between an input and events at other ports.
global scheduler	Def. 10	System that resolves non-determinism in PIOTSs: when applied to a PIOTS returns a PLTS.
localised scheduler	Def. 16	Same purpose as global schedulers but instead of a unique system, they are defined as a set of systems, one per each port. They are applied only to consistent PIOTSs.

Implementation relations		
Notation	*Definition*	*Explanation*
$r \equiv^G_{old} s$	Def. 4	A PLTS r is correct with respect to another PLTS s if for all trace σ of s both processes return the same probability for $[\sigma]$.
$r \sqsubseteq^G s$	Def. 5	Similar to $\equiv^G_{old}$ but considering only traces that end in quiescence.
$r \equiv^G s$	Def. 5	When restricted to a *finitary* class of processes the previous relation is an equivalence.
$r \sqsubseteq s$	Def. 9	Adaption of $\sqsubseteq^G$ to deal with consistent PIOTSs.
$r \equiv^s_g s$	Def. 13	A PIOTS r is correct with respect to another PIOTS s if for all global scheduler, its application to each of the processes return PLTSs that are equivalent under $\equiv^G$. If we apply a scheduler to the SUT then the same scheduler must provide an equivalent process when applied to the specification.
$r \sqsubseteq^w_g s$	Def. 14	A PIOTS r is correct with respect to another PIOTS s if for all global scheduler $\mathcal{G}_r$ there exists a global 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 $\equiv^G$. If we apply a scheduler to the SUT then the specification can choose a (possibly different) scheduler to find an equivalent process.
$r \equiv^s_l s$	Def. 17	Similar to $\equiv^s_g$ but for localised schedulers applied to consistent PIOTSs.
$r \sqsubseteq^w_l s$	Def. 17	Similar to $\sqsubseteq^w_g$ but for localised schedulers applied to consistent PIOTSs.

3. Implementation relations for probabilistic labelled transition systems

In this section we introduce a new implementation relation for probabilistic labelled transition systems with distributed ports. In Section 4 we extend this to models in which we distinguish between input and output. First we define the type of models we consider.

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, 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 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 and we assume that $\delta \in \mathcal{A}ct$. 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 $\mathcal{A}ct$.*

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

We assume that PLTSs are *connected*: for each state q of the system there is a sequence of transitions that reaches q from the initial state. In terms of the classification of probabilistic models [GSS95] we use a generative interpretation of probabilities. That is, for each state of the system, the sum of the probabilities associated with its outgoing transitions is 1 if δ self-loops are added to deadlocked states. Let us note that all transitions have non-zero probability[3]. We also do not allow two different transitions (q, a, q', p) and (q, a, q', p'): such a situation is equivalent to having a unique transition $(q, a, q', p + p')$. Finally, in the graphical representations of systems we omit probabilities equal to 1. We now introduce notation for PLTSs.

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,*

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

We say that $\sigma \in \mathcal{A}ct^*$ is a trace *of s if $prob(q_{in}, \sigma) > 0$. We denote by $L(s)$ the set of traces of s.*

It is clear that *prob* does not induce a probability measure on the set of traces. Nevertheless, the following result shows that *prob* is well-defined in the sense that given a fixed length n and a state q, *prob* induces a probability measure on the set of traces having length equal to n and outgoing from q.

Proposition 1 *Let $s = (Q, \mathcal{A}ct, T, q_{in})$ be a PLTS. For all $q \in Q$ and natural number $n \in \mathbb{N}$, consider the function $P_{q,n} : \mathcal{P}(\mathcal{A}ct^n) \longrightarrow [0,1]$ defined as $P_{q,n}(A) = \sum_{\sigma \in A} prob(q, \sigma)$. Then, $(\mathcal{A}ct^n, \mathcal{P}(\mathcal{A}ct^n), P_{q,n})$ is a probability space.*

Proof. We need to prove that $P_{q,n}$ is a probability measure. It is obvious that $P_{q,n}(\emptyset) = 0$ and that $P_{q,n}$ satisfies countable additivity. It remains to prove, by induction on n, that $P_{q,n}(\mathcal{A}ct^n) = 1$. The base case, $n = 0$, is trivial since $P_{q,n}(\mathcal{A}ct^0) = P_{q,n}(\{\epsilon\}) = prob(q, \epsilon) = 1$. For the inductive case, assume that the property holds for $n - 1$. We have that $P_{q,n}(\mathcal{A}ct^n) = P_{q,n}(\{a\sigma' \mid \exists p, q' : (q, a, q', p) \in T \wedge \sigma' \in \mathcal{A}ct^{n-1}\}) = \sum \{\!\!\{ prob(q, a\sigma') \mid \exists p, q' : (q, a, q', p) \in T \wedge \sigma' \in \mathcal{A}ct^{n-1} \}\!\!\}$, that is, it is enough to consider only traces beginning with an action that can be performed from q. Grouping terms and applying the definition of *prob* we have that the previous expression is equal to $\sum \{\!\!\{ p \cdot P_{q',n-1}(\mathcal{A}ct^{n-1}) \mid \exists p, q' : (q, a, q', p) \in T \}\!\!\}$. By the inductive hypothesis, for all state q' we have that $P_{q',n-1}(\mathcal{A}ct^{n-1}) = 1$ and taking into account that by the definition of PLTS, extended with δ transitions, the addition of all the probabilities labelling transitions departing each state of the system is equal to 1 we finally conclude that $P_{q,n}(\mathcal{A}ct^n) = 1$. $\square$

In distributed testing we cannot distinguish between sequences that are equivalent under $\sim$. Thus, rather than use the probability of traces we consider the probability of equivalence classes of traces. Note that the term *prob* is overloaded.

[3] An alternative is to allow the probability of a transition to be from the set $[0, 1]$ but we can simply delete any transition with probability 0 since it does not affect the behaviour of the PLTS.

Definition 3 *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*

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

Previously we defined an implementation relation, that we will call $\equiv^G_{old}$, in the following way [HN10].

Definition 4 *Let s, r be PLTSs. We write $r \equiv^G_{old} s$ if for all $\sigma \in L(s)$ we have that $prob(s, [\sigma]) = prob(r, [\sigma])$.*

This relation is an equivalence relation [HN10] and so the following result is clear.

Proposition 2 *Let s, r be PLTSs with action set $\mathcal{A}ct$. We have $r \equiv^G_{old} s$ if for all $\sigma \in \mathcal{A}ct^*$ we have that $prob(s, [\sigma]) = prob(r, [\sigma])$.*

This previously defined implementation relation allows us to distinguish between processes r and s such that r can do a_1a_2 and then deadlock (a_1 and a_2 are at different ports) and s can do a_2a_1 and then deadlock. All probabilities are 1 and r and s are not related under $\equiv^G_{old}$ since, for example, r can do a_1, and therefore $prob(r, [a_1]) = 1$ while s cannot, and therefore $prob(s, [a_1]) = 0$. However, as explained earlier, in distributed testing we can only bring together the observations at the separate ports when the system is quiescent since we need to know that the local traces observed are projections of the same global trace. As a result, often we will not be able to distinguish between processes such as r and s. Alternative approaches consist in synchronising the actions of the testers through the exchange of coordination messages [CR99, RC03, Hie12] or in having the individual testers exchanging messages with a central coordinator that records the global order of events and determines when an individual tester should send an input [JJKV98]. While these approaches can be extremely useful, they can complicate testing and are not always appropriate. Among the problems of these solutions we can mention that the time that it takes for the coordination messages to be received can distort the global order of events, that an implementation relation that implicitly assumes coordination messages might consider an SUT to be faulty even though it will appear to be correct in use, and that coordination messages are often sent using a network shared with the SUT and this might disturb the performance of the system.

If we restrict observations to traces that end in quiescent states then we obtain the following new implementation relation that we use in this paper.

Definition 5 *Let s, r be PLTSs with the same set $\mathcal{A}ct$ of actions. We write $r \sqsubseteq^G s$ 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 see that if $L(r)$ and $L(s)$ are finite then we have that $r \sqsubseteq^G s$ if and only if $s \sqsubseteq^G r$ and later (Proposition 3) we prove a slightly stronger result. When we use $\sqsubseteq^G$ with this type of *finitary* processes we will sometimes use the symbol $\equiv^G$ to make it clear that it is an equivalence relation. Note that this property does not hold if the processes have infinite sets of traces. 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 $r \sqsubseteq^G s$, since s does not have traces ending with quiescence, but the reverse relation, that is, $s \sqsubseteq^G r$, does not necessarily hold.

4. Probabilistic Input Output Transition Systems

Many systems interact with their environment through inputs and outputs and we now consider such systems and the observations that can be made. There is often an asymmetry between input and output since the environment controls the inputs while the system controls the outputs and this has led to the use of input output transition systems (IOTSs), which are LTSs where we distinguish between input and output. We now define a probabilistic IOTS that has multiple ports. In contrast to the purely generative model used in the previous section, we use the reactive scenario for inputs and the generative for outputs. We have a reactive scenario for inputs since the environment controls these but we still have a generative scenario for outputs since the SUT controls these. We attach probabilities to inputs since there may be more than one transition leaving a state q with a given input $?i$: the environment chooses the input to supply but the system determines which transition to take.

Definition 6 *A* probabilistic input-output transition system (PIOTS) $s = (Q, \texttt{In}, \texttt{Out}, T, q_{in})$ *is a tuple in which Q is a countable set of states, $q_{in} \in Q$ is the initial state,* $\texttt{In}$ *is a countable set of inputs,* $\texttt{Out}$ *is a countable set of outputs, and $T \subseteq Q \times (\texttt{In} \cup \texttt{Out}) \times Q \times (0,1]$ 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 $a \in \texttt{In} \cup \texttt{Out}$ with probability p. Again, we cannot have two transitions $(q, a, q', p) \in T$ and $(q, a, q', p') \in T$ in which $p \neq p'$. If $a \in \texttt{Out}$ 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 then with probability p this transition occurs. Therefore, for every state q we must have that $\sum \{\!| \, p \mid \exists q', a : (q, a, q', p) \in T \wedge a \in \texttt{Out} \, |\!\}$ is either 1 or 0 (if the state cannot produce any output). Further, if $a \in \texttt{In}$ 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 either 1 or 0 (if the input is not available at that state). This means that once an available 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 is no outgoing transition from q labelled by an output. We can 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 let $\mathcal{A}ct = \texttt{In} \cup \texttt{Out} \cup \{\delta\}$ denote the set of actions.*

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

We say that the process s is output-divergent *if it can reach a state in which there is an infinite path that contains outputs only. In this paper we only consider processes that are not output-divergent.*

As with PLTSs, we assume that PIOTSs are connected. If we treat δ as a normal output, then the sum of the probabilities associated with *outputs* from a state is always equal to 1, that is, for all $q \in Q$ we have $\sum \{\!| \, p \mid \exists q', a : (q, a, q', p) \in T_\delta \wedge a \notin \texttt{In} \, |\!\} = 1$. As usual, we precede the name of an input by ? and we precede the name of an output by !. We will often label inputs and outputs in order to make their port clear. For example, $?i_1$ denotes an input at 1 and $!o_1$ denotes an output at 1. An alternative [HMN08b] is to allow outputs to be tuples of values but the formalism used in this paper has the advantage of simplifying the notation and analysis.

Traces are sequences of actions, possibly including quiescence, that are sometimes called *suspension traces.* In this paper we call them *global traces.*

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

1. *If $(q, a, q', p) \in T_\delta$, for $a \in \mathcal{A}ct$, then we write $q \xrightarrow{a} q'$ and $q \xrightarrow{a}$.*
2. *We write $q \overset{\sigma}{\Longrightarrow} q'$ for $\sigma = a_1 \ldots a_m \in \mathcal{A}ct^*$ if there exist $q_0, \ldots, q_m$, $q = q_0$, $q' = q_m$ 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'.*

Note that we have initially abstracted probabilistic information in the definition of a trace. This information will be incorporated when defining implementation relations.

In distributed testing we cannot distinguish between traces that are equivalent under $\sim$ and so it makes sense to assign probabilities to equivalence classes of global traces. Consider a process that can do $?i_1?i_2$ and terminate or can do $?i_2?i_1$ and terminate. Each trace has associated probability 1 so if we sum the probabilities of the traces in $[?i_1?i_2]$ we obtain 2. This is because these traces are alternative outcomes of a race but the model does not contain probabilistic information regarding this race. As a result, in order to define an appropriate probabilistic extension of **dioco** [HMN08b, HMN08a, HMN12], an implementation relation to deal with systems with distributed ports, a previous approach [HN10] restricted attention to a class of systems without such *pathological* behaviours. Essentially, it did not consider systems with races between an input at a port o and events at other ports. The problem was that, as seen above, these systems do not induce a unique probability distribution function among its set of possible traces. Other work on using IOTSs for distributed systems defined the **mioco** implementation relation where global observations are made but there are multiple ports [BHT98]. This insists that if no transition is defined for input $?i_o \in \texttt{In}_o$

in state q, then there are no transitions from q with input from $\mathtt{In}_o$. This corresponds to the SUT being able to block input at an interface. We made a similar assumption, which is that for any state q we cannot have transitions from q for input at $o \in \mathcal{O}$ and also transitions with actions from other ports. This corresponds to a design that avoids races between an input and events at other ports. Interestingly, work on Message Sequence Charts [BAL97] has defined a pathology in which the next events after branching are on different processes: our restriction is similar to outlawing this pathology. We considered *consistent* systems as introduced below.

Definition 8 *A PIOTS* $(Q, \mathtt{In}, \mathtt{Out}, T, q_{in})$ *is* consistent *if for every state* $q \in Q$ *if there exist* $a_1, a_2 \in \mathtt{In} \cup \mathtt{Out}$ *such that* $q \xrightarrow{a_1}$ *and* $q \xrightarrow{a_2}$ *then either both of them are outputs or they are at the same port.*

Next, we introduce our previously defined implementation relation to relate consistent systems [HN10]. The idea is that an implementation r is correct with respect to specification s if for all quiescent trace σ of s we have that $[\sigma]$ has the same probability in both systems.

Definition 9 *Let* $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ *be consistent. We write* $r \sqsubseteq s$ *if for every sequence* σ *such that* $s \overset{\sigma\delta}{\Longrightarrow}$*, we have that* $prob(s, [\sigma\delta]) = prob(r, [\sigma\delta])$*.*

This definition is an extension of Definition 5 to consider consistent PIOTSs while in the previous definition we dealt with PLTS, the purely generative model without distinction between inputs and outputs. Definition 9 includes δ in the global traces to ensure that we are considering quiescent traces in each process. As discussed earlier, we consider quiescent traces since quiescence allows the testers to know that they are reporting projections of the same trace.

5. Global schedulers in the distributed architecture

In this section we show how schedulers can help us to deal with *pathological* processes since schedulers can be used to resolve some of the nondeterministic choices that complicate the computation of the probability associated with (classes of) sequences. Essentially, schedulers will be used to model potential *environments* for a process. This will allow us to define implementation relations for PIOTSs that are not consistent systems and thus generalise our previous implementation relations. In a certain sense, the objective of a scheduler is to transform a general reactive-generative process into a *tractable* process. Such schedulers fully quantify the choices of the system to which they are applied by producing a generative process.

First we consider schedulers that operate at the state level. The idea is that the next allowed action depends upon the current state of the process. These schedulers indicate, for each state q of the original PIOTS, whether we are choosing a specific port to both receive inputs and produce outputs or we are interested only in producing outputs. In the first case, probabilities have to be normalised to remove the weight previously assigned to outputs that will be discarded. The application of such a scheduler to a PIOTS produces a consistent system, and processes can be compared with our original implementation relation. However, consider processes u_1 and v_1 in Figure 1. Clearly, u_1 and v_1 should be considered to be equivalent since we can form u_1 from v_1 by unfolding the self-loop transition in state v_1 once. However, we can form a scheduler for u_1 in which we allow the transition from u_1 with input $?i_1$ but not that from u'' and we cannot simulate this in v_1 since there is only one transition with label $?i_1$. Therefore, while this notion of schedulers is conceptually appealing using them can be problematic.

We consider an alternative in which the behaviour of the scheduler depends on the global trace that has occurred. As a result, these schedulers are tree-like structures that indicate which actions can be performed at each point of time. Specifically, these schedulers are tree-like IOTSs with restrictions on the transitions departing from each state. The aim is for the composition of a scheduler with a process to be purely generative: we can then compose schedulers with the implementation and specification processes and compare the resultant generative processes using our new implementation relation $\equiv^G$.

In order to produce a generative process we require that a scheduler, when composed with a process, either can provide an input or it can wait and observe an output: we cannot have a race between an input and outputs. One option, to ensuring this, is to allow the scheduler to block output from the process. However, this does not fit too well with testing since often the environment cannot block output. In addition, if the schedulers can block output then they can distinguish between traces such as $!o_1!o_2$ and $!o_2!o_1$, by blocking the second output, despite such traces being observationally equivalent in the distributed setting.

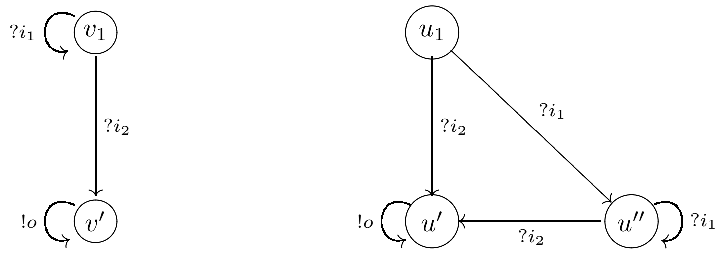

Fig. 1. Processes v_1 and u_1 that could be distinguished with a scheduler operating at the state level.

Instead of assuming that a scheduler can block output we assume that if an input is supplied then this is received before output is produced: if a process can produce output and the scheduler intends to send an input then the input wins the race. This assumption is less restrictive than requiring a scheduler to be able to block output: if a scheduler can block output then it can satisfy the requirement that an input will be received before output is produced by blocking all output whenever it is ready to supply input.

We now define (global) schedulers and then define the composition rules, for a process and a scheduler, that ensure that input is received before output is sent. In order to ensure that schedulers have a finite interaction with a process, and given that they have a tree-like structure, we only need to ask that their sets of states are finite. In particular, each trace generated by the application of a scheduler to a system will have a finite number of inputs, a property used in previous work on **dioco** to avoid pathological behaviours [HMN12].

Definition 10 *Let* `In` *and* `Out` *be sets of inputs and outputs, respectively, and $\mathcal{O}$ be a set of ports. A* global scheduler *for* `In` *and* `Out` *is a tuple $\mathcal{G} = (Q', \texttt{In}, \texttt{Out}, T', q'_{in})$ such that Q' is a finite set of states, with $q'_{in} \in Q'$ its initial state, and $T' \subseteq Q' \times \mathcal{A}ct \times Q'$ is 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 outputs and δ.*
- *For all $q \in Q'$, one of the following possibilities holds:*
  - *There exists* at most *one $a \in \texttt{In}$ and $q' \neq q$ such that $(q, a, q') \in T'$ and for all $b \in \texttt{Out} \cup \{\delta\}$, 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 and q' are pairwise different. These are the only transitions leaving q.*
  - *The only outgoing transitions are self-loops labelled by each action belonging to $\texttt{Out} \cup \{\delta\}$. 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 $L(\mathcal{G}) = \{\sigma \in (\texttt{In} \cup \texttt{Out} \cup \{\delta\})^* | \exists q' \in Q' : q'_{in} \stackrel{\sigma}{\Longrightarrow} q'\}$.*

Intuitively, after a trace is observed a scheduler can choose to supply an input and possibly observe output if the process cannot receive this input. Alternatively, it can choose to simply wait and observe an output or quiescence. In the next definition we introduce a construction to generate a global scheduler from a trace; this construction will be used in several proofs. The idea is that this trace is the *spine* of the produced global scheduler.

Definition 11 *Let* `In` *and* `Out` *be sets of inputs and outputs, respectively, $\mathcal{O}$ be a set of ports, and $\sigma \in \mathcal{A}ct^*$. The* global scheduler generated by *σ, denoted by $SG(\sigma)$, is a global scheduler $\mathcal{G} = (Q, \texttt{In}, \texttt{Out}, T, q_{in})$ such that (Q, T) are inductively constructed from the initial call $SG'(\sigma, q_{in}, \{q_{in}\}, \emptyset)$, as follows:*

$$SG'(\sigma_{aux}, q, Q_{aux}, T_{aux}) = \begin{cases} (Q_{aux}, T_{aux} \cup \{(q,a,q) | a \in \texttt{Out} \cup \{\delta\}\}) & \text{if } \sigma_{aux} = \epsilon \\ SG'(\sigma'_{aux}, q', Q_1, T_1) & \text{if } \sigma_{aux} = x\sigma'_{aux} \wedge x \in \mathcal{A}ct \end{cases}$$

where the state q' is fresh and Q_1 and T_1 are defined as follows:

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

Next we define the application of a global scheduler to a system.

Definition 12 *Let $s = (Q, \mathtt{In}, \mathtt{Out}, T, q_{in})$ be a PIOTS with a set of ports $\mathcal{O}$ and $\mathcal{G} = (Q', \mathtt{In}, \mathtt{Out}, T', q'_{in})$ be a global 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 one of the following holds.*

1. *$a \in \mathtt{In}$, $(q_1, a, q_2, p) \in T$ and $(q'_1, a, q'_2) \in T'$.*
2. *$a \in \mathtt{Out} \cup \{\delta\}$, $(q_1, a, q_2, p) \in T$, $(q'_1, a, q'_2) \in T'$ and there is no input $?i \in \mathtt{In}$, q_3, q'_3, and p' such that $(q_1, ?i, q_3, p') \in T$ and $(q'_1, ?i, q'_3) \in T'$.*

Of the two cases in the definition, the first simply represents the case where the environment and the SUT interact via an input being sent by the environment and the SUT being ready to receive this. The second is the case where the SUT is in a state where it can produce an output: as explained above, this only happens if the environment is not ready to send an input since in such situations the input wins the race.

It is straightforward to check that the application of global schedulers to PIOTSs produces processes similar to PLTSs but with the particularity that δ can be followed by observations other than δ. However, as with **dioco**, the tester (or environment) can choose to stop testing in a quiescent state and so the possible observations are projections of quiescent traces. Further, since we consider PIOTSs that are not output-divergent, the composition of a global scheduler and a process defines a PLTS with only finitely many traces that do not end in δ. The implementation relation $\equiv^G$ is therefore suitable. The next result makes use of this property to show that the application of global schedulers to processes keeps the symmetry of the $\sqsubseteq^G$ relation.

Proposition 3 *Let us suppose that $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ and $\mathcal{G}_r$ and $\mathcal{G}_s$ are global schedulers for* In *and* Out*. Then we have that $r \parallel \mathcal{G}_r \sqsubseteq^G s \parallel \mathcal{G}_s$ if and only if $s \parallel \mathcal{G}_s \sqsubseteq^G r \parallel \mathcal{G}_r$.*

Proof. We will assume that $r \parallel \mathcal{G}_r \sqsubseteq^G s \parallel \mathcal{G}_s$ and are required to prove that for every σ such that $\sigma\delta$ is a trace of r we have that $prob(r \parallel \mathcal{G}_r, [\sigma\delta]) = prob(s \parallel \mathcal{G}_s, [\sigma\delta])$. Define sets X_1 and X_2 of traces that do not end in δ such that $L(s \parallel \mathcal{G}_s)$ is the set of prefixes of $X_1\{\delta\}^*$ and $L(r \parallel \mathcal{G}_r)$ is the set of prefixes of $X_2\{\delta\}^*$. Since schedulers only apply finitely many inputs and processes are not output-divergent, X_1 and X_2 are finite. Let $[\sigma_1], \ldots, [\sigma_n]$ be the equivalence classes of maximal traces in X_2 and let $[\sigma'_1], \ldots, [\sigma'_m]$ be the equivalence classes of the maximal traces in X_1.

Since $r \parallel \mathcal{G}_r \sqsubseteq^G s \parallel \mathcal{G}_s$, for all $1 \leq i \leq m$ and $k \geq 1$ we have that $prob(r \parallel \mathcal{G}_r, [\sigma'_i\delta^k]) = prob(s \parallel \mathcal{G}_s, [\sigma'_i\delta^k])$ and so $\{[\sigma'_1], \ldots, [\sigma'_m]\} \subseteq \{[\sigma_1], \ldots, [\sigma_n]\}$. But, $\sum_{i=1}^{n} prob(r \parallel \mathcal{G}_r, [\sigma_i]) = 1$ and $\sum_{i=1}^{m} prob(s, [\sigma'_i]) = 1$ so $\{[\sigma'_1], \ldots, [\sigma'_m]\} = \{[\sigma_1], \ldots, [\sigma_n]\}$. Thus, if $\sigma\delta$ is a prefix of a sequence in X_2 then $prob(r \parallel \mathcal{G}_r, [\sigma\delta]) = prob(s, [\sigma\delta])$ as required. The result now follows from observing that for all σ_i and $k \geq 0$, $prob(r \parallel \mathcal{G}_r, [\sigma_i\delta^k]) = prob(r, [\sigma_i\delta])$ and $prob(s \parallel \mathcal{G}_s, [\sigma_i\delta^k]) = prob(s, [\sigma_i\delta])$. $\square$

We now define implementation relations using schedulers and $\equiv^G$. The following requires that for any choice of scheduler, we must have that the PLTSs that result from combining the scheduler with the implementation and specification PIOTSs are related under $\equiv^G$.

Definition 13 *Given $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, we write $r \equiv^s_g s$ if for all $\mathcal{G}$, global scheduler for* In *and* Out*, we have $r \parallel \mathcal{G} \equiv^G s \parallel \mathcal{G}$.*

The relation $\equiv^s_g$ requires that we compare the PLTSs produced when composing r and s with the *same* scheduler and to check this we effectively have to know how the environment behaves. However, this may be difficult especially when we have systems with physically distributed ports. We obtain a different implementation relation when the environment need not be known: all we can check is that for the given environment/global scheduler, the implementation behaves in a manner that is consistent with the specification for some (possibly different) environment/global scheduler. This provides the following 'weak' implementation relation, in contrast to the previous 'strong' implementation relation.

Definition 14 *Given $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, we write $r \sqsubseteq^w_g s$ if for all $\mathcal{G}_r$, global scheduler for* In *and* Out*, there exists $\mathcal{G}_s$, global scheduler for* In *and* Out*, such that $r \parallel \mathcal{G}_r \equiv^G s \parallel \mathcal{G}_s$.*

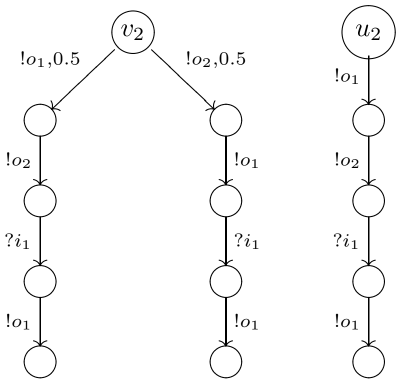

Fig. 2. Processes v_2 and u_2 where $u_2 \sqsubseteq_g^w v_2$ holds but $v_2 \sqsubseteq_g^w u_2$ does not.

The two alternative implementation relations, weak and strong, represent extremes. In the strong case we allow the possibility that the global behaviour of the environment is known. Typically, this is not feasible in a distributed environment but this implementation relation is 'safe' in situations in which there can be *some* knowledge regarding the global environment. For example, the outcome of what appears to be a race involving inputs sent to different ports may be known due to interactions between the agents at these ports. In contrast, the weak form represents the case where we know nothing about the environment. Thus, the choice of which implementation relation to use depends upon the context in which the SUT will be used and whether information regarding the global environment is available to users: if some such information is available then $\equiv_g^s$ should be used but otherwise the weaker relation, $\sqsubseteq_g^w$, can be used. Unlike $\equiv_g^s$, $\sqsubseteq_g^w$ is not an equivalence relation.

Proposition 4 *There exist $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq_g^w s$ but s does not conform to r under $\sqsubseteq_g^w$.*

Proof. Consider the processes $s = v_2$ and $r = u_2$ shown in Figure 2. First we show that $r \sqsubseteq_g^w s$. There are only two types of global schedulers to consider for r: one that sends input $?i_1$ after $!o_1!o_2$ and one that does not. In the first case we use a global scheduler $\mathcal{G}_s$ that sends input $?i_1$ after both $!o_1!o_2$ and $!o_2!o_1$ and in the second case we use a global scheduler $\mathcal{G}_s$ that does not send input. Thus, we have that $r \sqsubseteq_g^w s$ as required.

To see that s does not conform to r under $\sqsubseteq_g^w$ it is sufficient to consider global scheduler $\mathcal{G}_s$ that sends input $?i_1$ after $!o_1!o_2$ but not after $!o_2!o_1$. This gives probability 0.5 to $[!o_1!o_2?i_1!o_1\delta]$ in $s \parallel \mathcal{G}_s$ and clearly there is no $\mathcal{G}_r$ that gives the same probability for $[!o_1!o_2?i_1!o_1\delta]$ when composed with r. □

The next result follows immediately from the definition of a global scheduler.

Proposition 5 *Given $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ with empty sets of inputs, the following are equivalent statements:*

1. $r \equiv_g^s s$
2. $r \sqsubseteq_g^w s$
3. $r \sqsubseteq s$.

Proof. This follows from observing that since there are no inputs, there is only one possible global scheduler and the application of this scheduler to a process simply observes outputs until the process with which it is interacting reaches a quiescent state. □

We can now compare the implementation relations. Since $\sqsubseteq$ is defined only for consistent processes, the results will only refer to these. The following result is an immediate consequence of the definitions.

Proposition 6 *Given $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, if $r \equiv_g^s s$ then we also have that $r \sqsubseteq_g^w s$.*

However, the converse is not the case and this shows that the new implementation relations for PIOTSs differ even when we restrict attention to consistent processes.

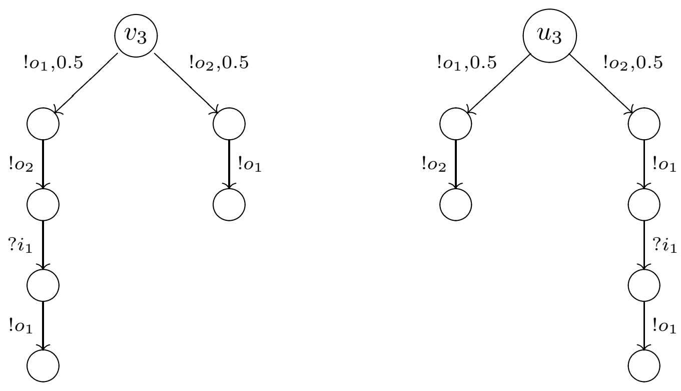

Fig. 3. Processes v_3 and u_3 where $u_3 \sqsubseteq^w_g v_3$ holds but $u_3 \equiv^s_g v_3$ does not.

Proposition 7 *There are consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq^w_g s$ holds but $r \equiv^s_g s$ does not. In addition, there are consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq^w_g s$ and $s \sqsubseteq^w_g r$ hold but $r \equiv^s_g s$ does not.*

Proof. Consider processes $r = u_3$ and $s = v_3$ shown in Figure 3. There are two types of relevant schedulers that can be applied to r: those that apply input $?i_1$ after $!o_2!o_1$ and those that do not. Given a scheduler $\mathcal{G}_r$ of the first type, we can obtain the same probabilities for the equivalence classes in s by choosing a scheduler that applies $?i_1$ after $!o_1!o_2$. Given a scheduler $\mathcal{G}_r$ of the second type, we can obtain the same probabilities for the equivalence classes in s by choosing a scheduler that does not apply input. Therefore we have that $r \sqsubseteq^w_g s$. Similarly, $s \sqsubseteq^w_g r$.

To see that $r \equiv^s_g s$ does not hold it is sufficient to choose any scheduler $\mathcal{G}$ that applies $?i_1$ after $!o_1!o_2$ but not after $!o_2!o_1$. We have that $0 = \mathit{prob}(r \parallel \mathcal{G}, [!o_1!o_2?i_1!o_1\delta]) \neq \mathit{prob}(s \parallel \mathcal{G}, [!o_1!o_2?i_1!o_1\delta]) = 0.5$. The result therefore holds. □

The following shows that the stronger of the new implementation relations is at least as strong as $\sqsubseteq$ when considering consistent processes.

Proposition 8 *Given consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, if $r \equiv^s_g s$ holds then $r \sqsubseteq s$ also holds.*

Proof. Assume that $r \equiv^s_g s$ and we will prove that this implies that $r \sqsubseteq s$. Since $r \equiv^s_g s$, we have that for all $\mathcal{G}_r$ we have that $r \parallel \mathcal{G}_r \equiv^G s \parallel \mathcal{G}_r$. We assume that σ is a trace that can take s to a quiescent state and it is sufficient to prove that $\mathit{prob}(s, [\sigma\delta]) = \mathit{prob}(r, [\sigma\delta])$.

If σ contains only outputs then we can choose a scheduler $\mathcal{G}_r$ that never applies input and we have that $\mathit{prob}(r \parallel \mathcal{G}_r, [\sigma\delta]) = \mathit{prob}(r, [\sigma\delta])$ and $\mathit{prob}(s \parallel \mathcal{G}_r, [\sigma\delta]) = \mathit{prob}(s, [\sigma\delta])$ and so the result holds. We therefore assume that σ contains at least one input.

Let $\sigma'_1, \ldots, \sigma'_k$ denote the longest prefixes of sequences from $[\sigma]$ that end in input. Further, let $c_1, \ldots, c_k$ be defined such that c_i is the set of traces in $[\sigma]$ that have prefix σ'_i. By definition, the c_i partition $[\sigma]$. Given σ'_i, we must have that $\mathit{prob}(r \parallel SG(\sigma'_i), [\sigma\delta])$ is the sum of the probabilities of traces from c_i in r. Since $r \equiv^s_g s$, we have that $\mathit{prob}(r \parallel SG(\sigma'_i), [\sigma\delta]) = \mathit{prob}(s \parallel SG(\sigma'_i), [\sigma\delta])$ for all $1 \leq i \leq k$. Since the c_i partition $[\sigma]$ we have that $\mathit{prob}(r, [\sigma\delta]) = \sum_{i=1}^{k} \mathit{prob}(r \parallel SG(\sigma'_i), [\sigma\delta])$ and $\mathit{prob}(s, [\sigma\delta]) = \sum_{i=1}^{k} \mathit{prob}(s \parallel SG(\sigma'_i), [\sigma\delta])$ and so $\mathit{prob}(s, [\sigma\delta]) = \mathit{prob}(r, [\sigma\delta])$ as required. □

Further, the new implementation relation $\equiv^s_g$ can be strictly stronger than $\sqsubseteq$ even if we consider the equivalence relation $\sqsubseteq \cap \sqsubseteq^{-1}$.

Proposition 9 *There are consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq s$ holds but $r \equiv^s_g s$ does not. In addition, there are consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq s$ and $s \sqsubseteq r$ hold but $r \equiv^s_g s$ does not.*

Proof. Again consider processes $r = u_3$ and $s = v_3$ shown in Figure 3. Clearly r and s are equivalent under $\sqsubseteq$ because $!o_1!o_2 \sim !o_2!o_1$. However, we saw in the proof of Proposition 7 that $r \equiv^s_g s$ does not hold. □

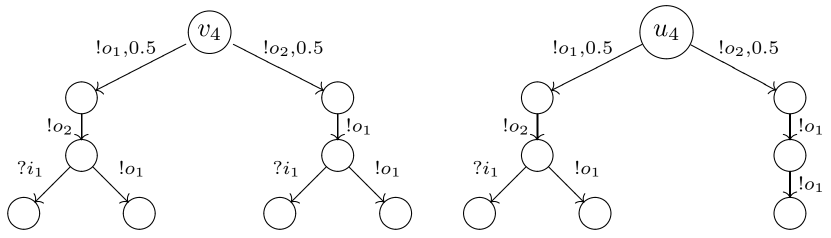

Fig. 4. Processes v_4 and u_4 where $u_4 \sqsubseteq_g^w v_4$ holds but $u_4 \sqsubseteq v_4$ does not.

Interestingly, the weaker of the new implementation relations need not be as strong as $\sqsubseteq$ when considering consistent processes.

Proposition 10 *There are consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq_g^w s$ holds but $r \sqsubseteq s$ does not.*

Proof. Consider processes $r = u_4$ and $s = v_4$ shown in Figure 4. We can observe that the quiescent traces of r and s are all in the equivalence classes $[!o_1!o_2!o_1\delta]$ and $[!o_1!o_2?i_1\delta]$. For r there is only one transition with an input and so there are only two types of schedulers to consider: those that apply $?i_1$ after $!o_1!o_2$ and those that do not provide input. Clearly, if we have a scheduler $\mathcal{G}_r$ that does not apply input then we obtain only one equivalence class, $[!o_1!o_2!o_1\delta]$, for both r and s and in each case we have probability 1. We can therefore use $\mathcal{G}_s = \mathcal{G}_r$. Now consider a scheduler $\mathcal{G}_r$ that applies $?i_1$ after $!o_1!o_2$. For r, both equivalence classes have probability 0.5. We obtain the same probabilities for the equivalence classes if, with s, we use a scheduler $\mathcal{G}_s$ that applies $?i_1$ after $!o_1!o_2$ but after no other sequences. Thus, we have that $r \sqsubseteq_g^w s$.

It is now sufficient to observe that we have that $1 = \mathit{prob}(s, [!o_1!o_2?i_1\delta]) \neq \mathit{prob}(r, [!o_1!o_2?i_1\delta]) = 0.5$ and so we do not have that $r \sqsubseteq s$. $\square$

In the proof of the above, in order to show that $r \sqsubseteq_g^w s$ we considered two types of schedulers and it might seem that in both cases we used the same scheduler for r and s. This could suggest that $r \equiv_g^s s$, providing a counterexample to Proposition 8. However, we can show that $r \equiv_g^s s$ does not hold by using a scheduler $\mathcal{G}$ that applies input $?i_1$ after both $!o_1!o_2$ and $!o_2!o_1$. Finally, we show that $\sqsubseteq$ is not stronger than $\sqsubseteq_g^w$.

Proposition 11 *There are consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq s$ holds but $r \sqsubseteq_g^w s$ does not.*

Proof. Consider processes $r = v_2$ and $s = u_2$ shown in Figure 2. Clearly r and s are equivalent under $\sqsubseteq$ because $!o_1!o_2 \sim !o_2!o_1$. However, as we showed in the proof of Proposition 4, $r \sqsubseteq_g^w s$ does not hold. $\square$

Overall, for consistent processes r and s, we have that $r \equiv_g^s s$ implies both that $r \sqsubseteq s$ and $r \sqsubseteq_g^w s$ but the converse directions do not hold. In addition, implementation relations $\sqsubseteq_g^w$ and $\sqsubseteq$ are incomparable.

Some of the above results might seem slightly surprising since we introduce schedulers in order to be able to define implementation relations for processes that are not consistent and so it seems natural to expect the implementation relations to be equivalent if we consider consistent processes. However, the proofs of Propositions 9, 10, and 11 use schedulers that behave differently after traces $!o_1!o_2$ and $!o_2!o_1$, despite these traces being observationally equivalent. Such schedulers can allow us to distinguish processes that cannot be distinguished by schedulers that always behave in the same way after any two traces σ and σ' such that $\sigma \sim \sigma'$. In some situations it may be reasonable to allow schedulers to behave differently after such σ and σ' since the environment might behave differently after observationally equivalent traces. For example, we might have that $?i_1$ is sent after $!o_1$ is observed and $?i_2$ is sent after $!o_2$ is observed and the order in which $!o_1$ and $!o_2$ are produced determines the order in which the inputs are received. In such a situation, the behaviour of the environment after $!o_1!o_2$ is different from the behaviour after $!o_2!o_1$: in the first case it next supplies input $?i_1$ and in the second case it next supplies input $?i_2$. However, there will be situations in which the environment will behave in the same way after any two traces σ and σ' with $\sigma \sim \sigma'$. For such situations we will want to restrict the class of schedulers considered and in the next section we show how this can be done.

6. Localised Schedulers

We have seen that a single global scheduler can be used to represent the environment of a system. However, if a system has physically distributed ports then it might interact with separate agents at these interfaces. Thus, in this section we consider an environment that contains a separate scheduler at each port.

The requirement that the agents at the separate interfaces are entirely independent might seem quite strong and clearly, by restricting the environment in this way we will obtain weaker implementation relations. However, the use of a weaker implementation relation has value, in providing a wider range of design choices, and so is useful in situations in which the environment is expected to behave in this way. For example, the system might interact with human users at its interfaces and it might be expected that these users will not interact with one another during the process of interacting with the system. Many web based systems have this property: each user has their own individual objective for using the system and this objective will not explicitly relate to the objectives of other users. If it is not possible to be confident that the environment will consist of separate independent agents then it may be safer to use an implementation relation based on global schedulers.

We call an environment that consists of separate agents a *localised scheduler*, since it is composed of independent distributed (local) schedulers.

Definition 15 *Let* $\mathtt{In}$ *and* $\mathtt{Out}$ *be sets of inputs and outputs, respectively, and* $\mathcal{O} = \{1, \ldots, m\}$ *be a set of ports such that the set* $\mathtt{In}$ *of inputs is partitioned into* $\mathtt{In}_1, \ldots, \mathtt{In}_m$ *and the set* $\mathtt{Out}$ *of outputs is partitioned into* $\mathtt{Out}_1, \ldots, \mathtt{Out}_m$*. A* localised scheduler $\mathcal{G}$ *for* $\mathtt{In}$ *and* $\mathtt{Out}$ *is defined by a tuple* $(\mathcal{G}_1, \ldots, \mathcal{G}_m)$ *where each* $\mathcal{G}_i$ *is a global scheduler with input set* $\mathtt{In}_i$ *and output set* $\mathtt{Out}_i$*. Given localised scheduler* $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_m)$*, we call each* $\mathcal{G}_i$ *a* local scheduler*. The* language *of* $\mathcal{G}$ *contains all the (finite) traces that can be performed by the scheduler. Therefore,* $L(\mathcal{G}) = \{\sigma \in \mathcal{A}ct^* | \forall o \in \mathcal{O}, \exists \sigma_o \in L(\mathcal{G}_o) : \sigma_o = \pi_o(\sigma)\}$*.*

Intuitively, at each point of time, a local scheduler of a localised scheduler can choose to supply an input $?i$ at its port, observing output or quiescence if the process cannot receive $?i$. Alternatively, it can choose to wait and observe an output or quiescence. We apply localised schedulers to consistent PIOTSs since otherwise the composition of s and $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_m)$ might be able to reach a point where more than one $\mathcal{G}_i$ can supply input. $\mathcal{G}$ does not define the probabilities of these different inputs, since the local schedulers act independently, and so we would obtain a race and would not be able to allocate associated probabilities. Note that $L(\mathcal{G})$ computes all the possible interleavings from different ports but since localised schedulers will be appplied only to consistent PIOTSs we will have that some of the sequences belonging to $L(\mathcal{G})$ will not be useful, in particular, if they cannot be performed by the PIOTS where the scheduler is being applied.

Definition 16 *Let* $s = (Q, \mathtt{In}, \mathtt{Out}, T, q_{in})$ *be a consistent PIOTS with port set* $\mathcal{O} = \{1, \ldots, m\}$ *and partitions of the sets of inputs and outputs into* $\mathtt{In}_1, \ldots, \mathtt{In}_m$ *and* $\mathtt{Out}_1, \ldots, \mathtt{Out}_m$*, respectively. Let* $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_m)$ *be a localised scheduler for* $\mathtt{In}$ *and* $\mathtt{Out}$ *such that for each* $o \in \mathcal{O}$*, we have* $\mathcal{G}_o = (Q_o, \mathtt{In}_o, \mathtt{Out}_o, T_o, q_{in}^o)$*. We define the application of* $\mathcal{G}$ *to* s*, denoted* $s \parallel \mathcal{G}$*, as the PLTS* $(Q', \mathcal{A}ct, T', (q_{in}, q_{in}^1, \ldots, q_{in}^m))$ *such that* $Q' \subseteq Q \times Q_1 \times \cdots \times Q_m$ *is the set of states reachable from the initial state under the set of transitions* T'*. We have that* $((q, q_1, \ldots, q_m), a, (q', q_1', \ldots, q_m'), p) \in T'$ *if and only if one of the following holds.*

1. $a \in \mathtt{In}_o$*,* $(q, a, q', p) \in T$*,* $(q_o, a, q_o') \in T_o$ *and for all* $o' \in \mathcal{O}$*, with* $o \neq o'$*, we have* $q_{o'} = q_{o'}'$*.*
2. $a \in \mathtt{Out}_o$*,* $(q, a, q', p) \in T$*,* $(q_o, a, q_o') \in T_o$*, for all* $o' \in \mathcal{O}$*, with* $o \neq o'$*, we have* $q_{o'} = q_{o'}'$*, and there are no* $o' \in \mathcal{O}$*, input* $?i \in \mathtt{In}_{o'}$*,* $q'', q_{o'}''$*, and* p' *such that* $(q, ?i, q'', p') \in T$ *and* $(q_{o'}, ?i, q_{o'}'') \in T_{o'}$*.*
3. $a = \delta$*,* $p = 1$*,* $(q, \delta, q, 1) \in T$*,* $q' = q$*, for all* $o \in \mathcal{O}$*,* $(q_o, \delta, q_o') \in T_o$*, and there exist no port* $o \in \mathcal{O}$*, input* $?i \in \mathtt{In}_o$*,* q'', q_o''*, and* p' *such that* $(q, ?i, q'', p') \in T$ *and* $(q_o, ?i, q_o'') \in T_o$*.*

It is possible to define two new implementation relations similar to the ones introduced previously.

Definition 17 *Given consistent* $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ *we write* $r \equiv_l^s s$ *if for all* $\mathcal{G}$*, localised scheduler for* $\mathtt{In}$ *and* $\mathtt{Out}$*, we have* $r \parallel \mathcal{G} \equiv^G s \parallel \mathcal{G}$*. Further, we write* $r \sqsubseteq_l^w s$ *if for all* $\mathcal{G}_r$*, localised scheduler for* $\mathtt{In}$ *and* $\mathtt{Out}$*, there exists* $\mathcal{G}_s$*, localised scheduler for* $\mathtt{In}$ *and* $\mathtt{Out}$*, such that* $r \parallel \mathcal{G}_r \equiv^G s \parallel \mathcal{G}_s$*.*

The distinction between $\equiv_l^s$ and $\sqsubseteq_l^w$ is similar to that for $\equiv_g^s$ and $\sqsubseteq_g^w$ described earlier. First, it is easy to see that $\equiv_l^s$ can be weaker than $\equiv_g^s$.

Proposition 12 *There are consistent PIOTSs $s, r \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \equiv_l^s s$ holds but $r \equiv_g^s s$ does not.*

Proof. Consider again processes $r = u_3$ and $s = v_3$ shown in Figure 3. We have previously shown, in the proof of Proposition 7, that $r \equiv_g^s s$ does not hold. However, it is not possible to find a localised scheduler that distinguishes between these processes since a localised scheduler must behaved in the same way after $!o_1!o_2$ and $!o_2!o_1$. $\square$

The following result will be useful in reasoning about relations $\equiv_l^s$ and $\sqsubseteq_l^w$.

Proposition 13 *Let us suppose that for localised scheduler $\mathcal{G}$ we have that $[\sigma] \cap L(\mathcal{G}) \neq \emptyset$. Then for any consistent process r we have that $prob(r, [\sigma]) = prob(r \parallel \mathcal{G}, [\sigma])$.*

Proof. We will assume that $[\sigma] \cap L(\mathcal{G}) \neq \emptyset$ for localised scheduler $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_m)$. Thus, there exists $\sigma' \sim \sigma$ such that $\sigma' \in L(\mathcal{G})$ and so $\pi_o(\sigma') \in L(\mathcal{G}_o)$ for all $o \in \mathcal{O}$. Since $\sigma' \sim \sigma$, $\pi_o(\sigma) = \pi_o(\sigma')$ and so for all $o \in \mathcal{O}$ we have that $\pi_o(\sigma) \in L(\mathcal{G}_o)$.

Now consider some $\sigma' \sim \sigma$. Since for all $o \in \mathcal{O}$ we have that $\pi_o(\sigma') = \pi_o(\sigma)$ and $\pi_o(\sigma) \in L(\mathcal{G}_o)$, if σ' is a trace of r then $r \parallel \mathcal{G}$ can perform σ' unless an input supplied by some $\mathcal{G}_o$ beats an output from σ' at a port $o' \neq o$ in a race. However, since r is consistent this is not possible. Thus, if σ' is a trace of r then it is a trace of $r \parallel \mathcal{G}$ and it has the same probability in each. The result therefore holds. $\square$

Actually, we have that $\equiv_l^s$ is strictly weaker than $\equiv_g^s$.

Proposition 14 *Given consistent $r, s \in \mathcal{PIOTS}(I, O)$, if $r \equiv_g^s s$ then $r \equiv_l^s s$.*

Proof. We assume that $r \equiv_l^s s$ does not hold and so there is a localised scheduler $\mathcal{G}$ and σ, with $s \xRightarrow{\sigma\delta}$, such that $prob(r \parallel \mathcal{G}, [\sigma\delta]) \neq prob(s \parallel \mathcal{G}, [\sigma\delta])$. It is sufficient to prove that $r \equiv_g^s s$ does not hold.

We now consider two cases. First, if σ contains no input then we choose the global scheduler $\mathcal{G}' = SG(\sigma)$ that applies no input. By Proposition 13 we have that $prob(r \parallel \mathcal{G}', [\sigma\delta]) = prob(r \parallel \mathcal{G}, [\sigma\delta])$ and $prob(s \parallel \mathcal{G}', [\sigma\delta]) = prob(s \parallel \mathcal{G}, [\sigma\delta])$. Thus, since $prob(r \parallel \mathcal{G}, [\sigma\delta]) \neq prob(s \parallel \mathcal{G}, [\sigma\delta])$ we have that $prob(r \parallel \mathcal{G}', [\sigma\delta]) \neq prob(s \parallel \mathcal{G}', [\sigma\delta])$ as required.

Now assume that σ contains one or more inputs. Let $\sigma_1, \ldots, \sigma_k$ denote longest prefixes of elements of $[\sigma]$ that end in input. For $1 \leq j \leq k$ let c_j denote the set of traces from $[\sigma]$ that have σ_j as a prefix and so the c_j partition $[\sigma]$. Since $prob(r \parallel \mathcal{G}, [\sigma\delta]) \neq prob(s \parallel \mathcal{G}, [\sigma\delta])$, at least one of these probabilities is non-zero and so we must have that $[\sigma] \cap L(\mathcal{G}) \neq \emptyset$. Thus, by Proposition 13, since $prob(r \parallel \mathcal{G}, [\sigma\delta]) \neq prob(s \parallel \mathcal{G}, [\sigma\delta])$ we have that $prob(r, [\sigma\delta]) \neq prob(s, [\sigma\delta])$. Thus, there exists $1 \leq i \leq k$ such that $\sum_{\sigma' \in c_i} prob(r, \sigma'\delta) \neq \sum_{\sigma' \in c_i} prob(s, \sigma'\delta)$. Consider global scheduler $\mathcal{G}' = SG(\sigma_i)$. Then $prob(r \parallel \mathcal{G}', [\sigma\delta]) = \sum_{\sigma' \in c_i} prob(r, \sigma'\delta)$ and $prob(s \parallel \mathcal{G}', [\sigma\delta]) = \sum_{\sigma' \in c_i} prob(s, \sigma'\delta)$. Thus, $prob(r \parallel \mathcal{G}', [\sigma\delta]) \neq prob(s \parallel \mathcal{G}', [\sigma\delta])$ as required. $\square$

We now compare $\equiv_l^s$ with $\sqsubseteq$; later we will show how $\sqsubseteq_l^w$ and $\sqsubseteq$ relate.

Proposition 15 *Given consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, if $r \equiv_l^s s$ then $r \sqsubseteq s$. However, there exist consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq s$ but $r \equiv_l^s s$ does not hold.*

Proof. First assume that $r \equiv_l^s s$ and we will prove that $r \sqsubseteq s$. Let σ be an arbitrary trace and it is sufficient to prove that $prob(r, [\sigma\delta]) = prob(s, [\sigma\delta])$. We will define a localised scheduler $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_m)$ in the following way: for all $o \in \mathcal{O}$ define $\mathcal{G}_o = SG(\pi_o(\sigma))$. Since $r \equiv_l^s s$, we have that $prob(r \parallel \mathcal{G}, [\sigma\delta]) = prob(s \parallel \mathcal{G}, [\sigma\delta])$. By construction we have that $[\sigma] \cap L(\mathcal{G}) \neq \emptyset$ and so by Proposition 13 we know that $prob(r, [\sigma\delta]) = prob(r \parallel \mathcal{G}, [\sigma\delta])$ and $prob(s, [\sigma\delta]) = prob(s \parallel \mathcal{G}, [\sigma\delta])$. Thus, $prob(r, [\sigma\delta]) = prob(s, [\sigma\delta])$ as required.

For the second part, consider $s = u_5$ and $r = v_5$ shown in Figure 5. It is straightforward to see that $r \sqsubseteq s$ since the only traces that we have to consider are the traces of s. However, to see that $r \equiv_l^s s$ does not hold we can use a scheduler that applies input $?i_1$ after $!o_1$. $\square$

We now compare $\sqsubseteq_g^w$ and $\sqsubseteq_l^w$. Interestingly, even though using localised schedulers is a restriction, $r \sqsubseteq_g^w s$ does not imply $r \sqsubseteq_l^w s$, since we also place restrictions on the scheduler $\mathcal{G}_s$ that can be used.

Proposition 16 *There are consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq_g^w s$ but we do not have that $r \sqsubseteq_l^w s$.*

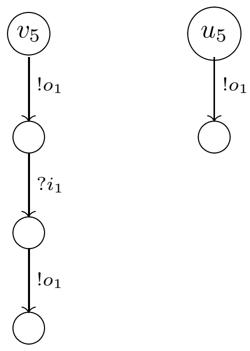

Fig. 5. Processes v_5 and u_5 where $u_5 \sqsubseteq_l^w v_5$ but not $u_5 \sqsubseteq v_5$.

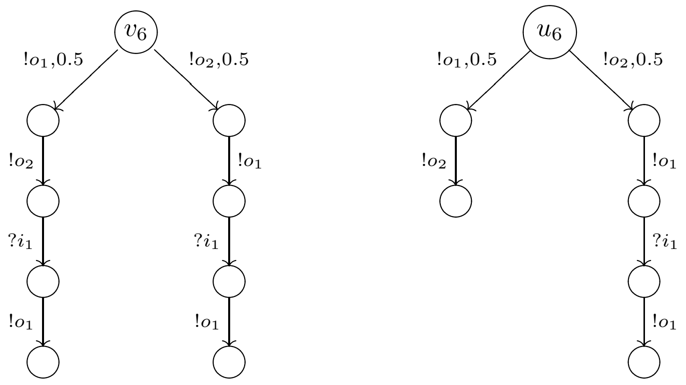

Fig. 6. Processes v_6 and u_6 where $u_6 \sqsubseteq_g^w v_6$ holds but $u_6 \sqsubseteq_l^w v_6$ does not.

Proof. Consider the processes $r = u_6$ and $s = v_6$ shown in Figure 6. It is clear that we cannot distinguish between these processes using a (global or localised) scheduler that does not apply input. For any global scheduler $\mathcal{G}_r$ that applies input $?i_1$ after $!o_2!o_1$ we can choose the global scheduler $\mathcal{G}_s$ that applies input $?i_1$ after $!o_2!o_1$ but otherwise does not apply input. Clearly $s \parallel \mathcal{G}_s$ and $r \parallel \mathcal{G}_r$ give the same probabilities to the possible traces and so $r \sqsubseteq_g^w s$. To see that r does not conform to s under $\sqsubseteq_l^w$ we can choose a scheduler $\mathcal{G}_r$ in which the local scheduler at port 2 does not apply input and the local scheduler at port 1 applies input $?i_1$ after $!o_1$. This gives $[!o_2!o_1?i_1!o_1]$ probability 0.5. Any localised scheduler $\mathcal{G}_s$ for s must behave in the same way after $!o_1!o_2$ and $!o_2!o_1$ and so must give $[!o_2!o_1?i_1!o_1]$ probability 0 or 1. Thus, r does not conform to s under $\sqsubseteq_l^w$ as required. □

Proposition 17 *There are consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ such that $r \sqsubseteq_l^w s$ but we do not have that $r \sqsubseteq_g^w s$.*

Proof. Consider the processes $r = v_2$ and $s = u_2$ in Figure 2. In the proof of Proposition 4 we have already seen that $r \sqsubseteq_g^w s$ does not hold. However, any localised scheduler $\mathcal{G}_r$ for r must behave in the same way after $!o_1!o_2$ and $!o_2!o_1$ and so cannot distinguish between these processes. □

Finally, $\equiv_l^s$ and $\sqsubseteq_l^w$ are related in the way one would expect.

Proposition 18 *Given consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, if $r \equiv_l^s s$ then $r \sqsubseteq_l^w s$ but it is possible that $r \sqsubseteq_l^w s$ and that $r \equiv_l^s s$ does not hold.*

Proof. The first part is immediate from the definition. For the second part, let r and s be u_5 and v_5 from Figure 5 respectively. We have that $r \sqsubseteq_l^w s$ since for any localised scheduler $\mathcal{G}_r$ we simply use the localised

Property	**Corresponding results**	**Property**	**Corresponding results**
$\equiv_g^s \prec \sqsubseteq_g^w$	Propositions 6 and 7	$\sqsubseteq_g^w \ominus \sqsubseteq$	Propositions 10 and 11
$\equiv_g^s \prec \sqsubseteq$	Propositions 8 and 9	$\equiv_l^s \prec \sqsubseteq$	Proposition 15
$\equiv_g^s \prec \equiv_l^s$	Propositions 12 and 14	$\sqsubseteq_g^w \ominus \sqsubseteq_l^w$	Propositions 16 and 17
$\equiv_l^s \prec \sqsubseteq_l^w$	Proposition 18	$\sqsubseteq_g^w \ominus \equiv_l^s$	Proposition 19
$\equiv_g^s \prec \sqsubseteq_l^w$	$\equiv_g^s \prec \equiv_l^s \wedge \equiv_l^s \prec \sqsubseteq_l^w$	$\sqsubseteq_l^w \ominus \sqsubseteq$	Proposition 20

$$\begin{tikzcd}
 & r \sqsubseteq_l^w s & \\
r \equiv_l^s s \arrow[ur] \arrow[dr] & r \equiv_g^s s \arrow[l] \arrow[r] & r \sqsubseteq_g^w s \\
 & r \sqsubseteq s &
\end{tikzcd}$$

Fig. 7. Comparing the implementation relations.

scheduler $\mathcal{G}_s$ for s that does not apply input: both $r \parallel \mathcal{G}_r$ and $s \parallel \mathcal{G}_s$ have only two traces ϵ and $!o_1$, both with probability 1. To see that r does not conform to s under $\equiv_l^s$ it is sufficient to use a localised scheduler $\mathcal{G}$ that applies input $?i_1$ after $!o_1$. □

To summarise, if we restrict attention to localised schedulers then, for consistent processes, the new implementation relation $\equiv_l^s$ is stronger than $\sqsubseteq_l^w$ and the implementation relation $\sqsubseteq$. In addition, we will see that $\sqsubseteq$ and $\sqsubseteq_l^w$ are incomparable. We now compare the relations $\sqsubseteq_g^w$ and $\equiv_l^s$.

Proposition 19 *Given consistent $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$, it is possible that $r \equiv_l^s s$ and that $r \sqsubseteq_g^w s$ does not hold and it is possible that $r \sqsubseteq_g^w s$ and that $r \equiv_l^s s$ does not hold.*

Proof. For the first part consider $r = v_2$ and $s = u_2$ shown in Figure 2. In the proof of Proposition 4 we have already seen that $r \sqsubseteq_g^w s$ does not hold. However, no localised scheduler can distinguish these processes. The proof of the second part is equivalent to the proof of the second part of Proposition 18. □

Finally, we compare $\sqsubseteq_l^w$ to $\sqsubseteq$.

Proposition 20 *Let us suppose that $r, s \in \mathcal{PIOTS}(\mathtt{In}, \mathtt{Out})$ are consistent. We have that $r \sqsubseteq s$ does not imply that $r \sqsubseteq_l^w s$ and $r \sqsubseteq_l^w s$ does not imply that $r \sqsubseteq s$.*

Proof. For the first part, consider $r = v_5$ and $s = u_5$ shown in Figure 5. Again, $r \sqsubseteq s$ since the only traces that we have to consider are the traces of s. However, to see that $r \sqsubseteq_l^w s$ does not hold we can use a scheduler $\mathcal{G}_r$ that applies input $?i_1$ after $!o_1$.

We now show that $r \sqsubseteq_l^w s$ does not imply that $r \sqsubseteq s$. Consider the processes $r = u_5$ and $s = v_5$ shown in Figure 5. Clearly, we do not have that $r \sqsubseteq s$ since s contains the trace $!o_1?i_1!o_1$ that is not in r. It is therefore sufficient to show that $r \sqsubseteq_l^w s$. However, for any localised scheduler $\mathcal{G}_r$ we have that $r \parallel \mathcal{G}_r$ has only two traces: ϵ and $!o_1$, both with probability 1. We obtain the same behaviour for $s \parallel \mathcal{G}_s$ if we choose $\mathcal{G}_s$ so that it does not provide input. Thus, $r \sqsubseteq_l^w s$ as required. □

Figure 7 summarises the relation between the different implementation relations presented in the paper, where *Corresponding Result* either quotes a proposition or lists a sequence of results that allows us to conclude that the corresponding property holds. In this table, given relations $\sqsubseteq_1$ and $\sqsubseteq_2$:

1. $\sqsubseteq_1 \prec \sqsubseteq_2$ if $r \sqsubseteq_1 s$ implies $r \sqsubseteq_2 s$ and also $r \sqsubseteq_2 s$ does not imply $r \sqsubseteq_1 s$.
2. $\sqsubseteq_1 \ominus \sqsubseteq_2$ if $r \sqsubseteq_1 s$ does not imply $r \sqsubseteq_2 s$ and also $r \sqsubseteq_2 s$ does not imply $r \sqsubseteq_1 s$.

Note that these results apply to consistent processes since some of the implementation relations are only defined for such processes. In the same figure, we present a graphical representation of the results table.

7. Conclusions and Future Work

This paper has considered the situation in which we have a probabilistic model of the SUT, the SUT interacts with its environment at physically distributed ports, and we place a separate tester at each port. The tester at a port o only observes events that occur at o and, as a result, in testing we observe the set of local projections of the global trace that occurred. This induces an equivalence relation $\sim$ on global traces: two global traces are equivalent under $\sim$ if they have the same local projections.

Previous work explored distributed testing from a probabilistic input output transition system (PIOTS) [HN10]. Since we can only observe traces up to $\sim$, it was necessary to consider probabilities associated with equivalence classes of global traces. This led to a problem: since we used a reactive-generative scheme for PIOTSs, the addition of the probabilities of global traces in an equivalence class need not be meaningful. As a result, the work only considered consistent processes in which it was not possible to have a transition with input at port o from a state q if there was another transition from q involving an event at a port $o' \neq o$. This restriction allowed implementation relation $\sqsubseteq$ to be defined.

In this paper we used schedulers to represent possible environments for the SUT. Importantly, the composition of a PIOTS and a scheduler is a generative process and so the use of schedulers allowed us to consider any PIOTSs. We defined two implementation relations for an SUT r and specification s: for the stronger relation $\equiv_g^s$ we required that for any choice of scheduler $\mathcal{G}$ we have that the composition of r and $\mathcal{G}$ is indistinguishable from the composition of s and $\mathcal{G}$. For the weaker implementation relation, $\sqsubseteq_g^w$, we relaxed this to consider the situation in which r interacts with an environment modelled by a scheduler $\mathcal{G}_r$ but we cannot know $\mathcal{G}_r$: for any choice of scheduler $\mathcal{G}_r$ there is a scheduler $\mathcal{G}_s$ (a possible environment) such that the composition of r and $\mathcal{G}_r$ is indistinguishable from the composition of s and $\mathcal{G}_s$. Clearly, we have that $\equiv_g^s$ is strictly stronger than $\sqsubseteq_g^w$. It transpired that if we restrict attention to consistent PIOTS then $\equiv_g^s$ is strictly stronger than both $\sqsubseteq_g^w$ and $\sqsubseteq$ and also that $\sqsubseteq_g^w$ and $\sqsubseteq$ are incomparable.

We observed that a global scheduler, representing the environment, might behave differently after two traces σ and σ' that are observationally equivalent. In some situations this might not be realistic and so we investigated an approach to eliminate such possibilities: we defined a localised scheduler that contains one scheduler for each port. This did not affect the relationships between the implementation relations corresponding to $\equiv_g^s$, $\sqsubseteq_g^w$, and $\sqsubseteq$.

There are still several avenues for future work on the topic of distributed testing of probabilistic systems. A first line that we would like to follow is to define a complete testing framework from a more *algebraic* point of view. The work by Carroll Morgan and colleagues on the topic [DGH$^+$07, DGHM08, DGHM09, DGHM11] is a very valuable initial step, in particular, because the ideas underlying the definition of alternative characterisations of the may and must preorders as simulation relations could be adapted to our framework.

If we consider global schedulers that behave as locally consistent ones, that is, such that the decision to apply an input at a port o depends only on the observations at o, then it is easy to transform them into *equivalent* localised schedulers. Consider, now, the problem of representing a localised scheduler using a global scheduler. We might have a local scheduler at port 1 that starts by applying input $?i_1$ and a local scheduler at port 2 that starts by applying input $?i_2$. In this situation there are two possible initial inputs, a situation that we cannot model with our global schedulers. This is not problematic if we apply the localised scheduler to a single consistent process, since there cannot be a race in the composition of the localised scheduler and the process: the composition results in a generative process as required. However, when we compare different consistent processes r and s, the compositions of r and s with a localised scheduler might resolve these races in different ways and so we cannot replace such a localised scheduler by a global scheduler. We believe that the use of *probabilistic schedulers*, which have probabilistic information regarding different choices (supplying a particular input or waiting for outputs), can partially alleviate this problem. However, the transition will not be automatic since probabilistic schedulers might require probabilistic information that cannot be extracted from localised schedulers. Another possibility, that we would also like to explore, is to adapt the notion of a scheduler that has been devised for model checking distributed systems [GD09]. We have explored two extremes: global schedulers and localised schedulers. An alternative would lie in the middle of them: to use a set of local schedulers and one scheduler to indicate which component is active. Looking for boundaries, our work on localised schedulers is restricted to consistent systems. This is a small class of systems and it might be interesting to find bigger sets of PIOTSs where localised schedulers can be still applied. In this direction it might be relevant to consider the class of controllable systems as studied in the non-probabilistic setting.

We have observed an interesting property: $r \sqsubseteq_g^w s$ does not imply $r \sqsubseteq s$. This is because we can choose the scheduler for s so that it avoids the *problematic* part of the process that would allow us to distinguish r and s. As an extreme example, the null process that remains in a quiescent state conforms to any process s that is initially quiescent under our weak relations: we choose a scheduler for s that does not supply input ($r \parallel \mathcal{G}_r$ and $s \parallel \mathcal{G}_s$ both have only the empty sequence). A possible solution, that we would like to explore in the future, consists of restricting the type of schedulers that s can use to *simulate* r in a weak relation. Even though the motivation for defining $\sqsubseteq_g^w$ is that sometimes we cannot know the environment, due to its distributed nature, we might know the local projections. Thus it would be interesting to analyse the resulting weak implementation relations if we require that the two schedulers have the same local projections.

Acknowledgements

We would like to thank the anonymous referees for the constructive reviews that have helped to further strengthen the paper.

References

[AB00] A. Aldini and M. Bravetti. An asynchronous calculus for generative-reactive probalistic systems. In *8th Process Algebras and Performance Modelling Workshop, PAPM'00*, pages 591–606. Carleton Scientific, 2000.

[AO08] P. Ammann and J. Offutt. *Introduction to Software Testing.* Cambridge University Press, 2008.

[BA03] M. Bravetti and A. Aldini. Discrete time generative-reactive probabilistic processes with different advancing speeds. *Theoretical Computer Science*, 290(1):355–406, 2003.

[BAL97] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in Message Sequence Charts. In *3rd Int. Workshop on Tools and Algorithms for Construction and Analysis of Systems, TACAS'97, LNCS 1217*, pages 259–274, 1997.

[BHT98] E. Brinksma, L. Heerink, and J. Tretmans. Factorized test generation for multi-input/output transition systems. In *11th IFIP Workshop on Testing of Communicating Systems, IWTCS'98*, pages 67–82. Kluwer Academic Publishers, 1998.

[BU91] S.C. Boyd and H. Ural. The synchronization problem in protocol testing and its complexity. *Information Processing Letters*, 40(3):131–136, 1991.

[CDSY99] R. Cleaveland, Z. Dayar, S.A. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93–148, 1999.

[Chr90] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In *1st Int. Conf. on Concurrency Theory, CONCUR'90, LNCS 458*, pages 126–140. Springer, 1990.

[CLSV06] L. Cheung, N. Lynch, R. Segala, and F. Vaandrager. Switched PIOA: Parallel composition via distributed scheduling. *Theoretical Computer Science*, 365(1-2):83–108, 2006.

[CR99] L. Cacciari and O. Rafiq. Controllability and observability in distributed testing. *Information and Software Technology*, 41(11–12):767–780, 1999.

[CSV07] L. Cheung, M. Stoelinga, and F. Vaandrager. A testing scenario for probabilistic processes. *Journal of the ACM*, 54(6):Article 29, 2007.

[DB85] R. Dssouli and G. von Bochmann. Error detection with multiple observers. In *5th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'85*, pages 483–494. North-Holland, 1985.

[DB86] R. Dssouli and G. von Bochmann. Conformance testing with multiple observers. In *6th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'86*, pages 217–229. North-Holland, 1986.

[DGH+07] Y. Deng, R. van Glabbeek, M. Hennessy, C. Morgan, and C. Zhang. Characterising testing preorders for finite probabilistic processes. In *22nd Annual IEEE Symposium on Logic in Computer Science, LICS'07*, pages 313–325. IEEE Computer Society, 2007.

[DGHM08] Y. Deng, R. van Glabbeek, M. Hennessy, and C. Morgan. Characterising testing preorders for finite probabilistic processes. *Logical Methods in Computer Science*, 4(4), 2008.

[DGHM09] Y. Deng, R. van Glabbeek, M. Hennessy, and C. Morgan. Testing finitary probabilistic processes. In *20th Int. Conf. on Concurrency Theory, CONCUR'09, LNCS 5710*, pages 274–288. Springer, 2009.

[DGHM11] Y. Deng, R. van Glabbeek, M. Hennessy, and C. Morgan. Real-reward testing for probabilistic processes (extended abstract). In *9th Workshop on Quantitative Aspects of Programming Languages, QAPL'11, EPTCS 57*, pages 61–73, 2011.

[Gau95] M.-C. Gaudel. Testing can be formal, too! In *6th Int. Joint Conf. CAAP/FASE, Theory and Practice of Software Development, TAPSOFT'95, LNCS 915*, pages 82–96. Springer, 1995.

[GD09] S. Giro and P.R. D'Argenio. On the expressive power of schedulers in distributed probabilistic systems. In *7th Workshop on Quantitative Aspects of Programming Languages, QAPL'09*, pages 45–71. Electronic Notes in Theoretical Computer Science 253(3), 2009.

[GGSV02] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. In *ACM SIGSOFT Symposium on Software Testing and Analysis, ISSTA'02*, pages 112–122. ACM Press, 2002.

[GKSB11] W. Grieskamp, N. Kicillof, K. Stobie, and V. Braberman. Model-based quality assurance of protocol documentation: tools and methodology. *Software Testing, Verification and Reliability*, 21(1):55–71, 2011.

[GSS95] R. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[HBB$^+$09] R. M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A.J.H Simons, S. Vilkomir, M.R. Woodward, and H. Zedan. Using formal methods to support testing. *ACM Computing Surveys*, 41(2), 2009.

[HBH08] R. M. Hierons, J.P. Bowen, and M. Harman, editors. *Formal Methods and Testing, LNCS 4949*. Springer, 2008.

[Hie12] R. M. Hierons. Overcoming controllability problems in distributed testing from an input output transition system. *Distributed Computing*, 25(1):63–81, 2012.

[HM09] R. M. Hierons and M. G. Merayo. Mutation testing from probabilistic and stochastic finite state machines. *Journal of Systems and Software*, 82(11):1804–1818, 2009.

[HMN08a] R. M. Hierons, M. G. Merayo, and M. Núñez. Controllable test cases for the distributed test architecture. In *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311*, pages 201–215. Springer, 2008.

[HMN08b] R. M. Hierons, M. G. Merayo, and 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*, pages 200–215. Springer, 2008.

[HMN12] R. M. Hierons, M. G. Merayo, and M. Núñez. Implementation relations and test generation for systems with distributed interfaces. *Distributed Computing*, 25(1):35–62, 2012.

[HN10] R. M. Hierons and M. Núñez. Testing probabilistic distributed systems. In *IFIP 30th Int. Conf. on Formal Techniques for Distributed Systems, FMOODS/FORTE'10, LNCS 6117*, pages 63–77. Springer, 2010.

[HU08] R. M. Hierons and H. Ural. The effect of the distributed test architecture on the power of testing. *The Computer Journal*, 51(4):497–510, 2008.

[JJKV98] C. Jard, T. Jéron, H. Kahlouche, and C. Viho. Towards automatic distribution of testers for distributed conformance testing. In *TC6 WG6.1 Joint Int. Conf. on Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE'98*, pages 353–368. Kluwer Academic Publishers, 1998.

[LDB93] G. Luo, R. Dssouli, and G. von Bochmann. Generating synchronizable test sequences based on finite state machine with distributed ports. In *6th IFIP Workshop on Protocol Test Systems, IWPTS'93*, pages 139–153. North-Holland, 1993.

[LNR06] N. López, M. Núñez, and I. Rodríguez. Specification, testing and implementation relations for symbolic-probabilistic systems. *Theoretical Computer Science*, 353(1–3):228–248, 2006.

[LS91] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[LV95] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations I: Untimed systems. *Information and Computation*, 121(2):214–233, 1995.

[MMS96] C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, 1996.

[MMSS96] C. Morgan, A. McIver, K. Seidel, and J. W. Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 8(6):617–647, 1996.

[Mor88] C. Morgan. The specification statement. *ACM Transactions on Programming Languages and Systems*, 10(3):403–419, 1988.

[Mor90] C. Morgan. *Programming from specifications*. Prentice Hall, 1990.

[Mye04] G.J. Myers. *The Art of Software Testing*. John Wiley and Sons, 2nd edition, 2004.

[Núñ03] M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.

[RC03] O. Rafiq and L. Cacciari. Coordination algorithm for distributed testing. *The Journal of Supercomputing*, 24(2):203–211, 2003.

[SB84] B. Sarikaya and G. von Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, 32:389–395, 1984.

[Seg95] R. Segala. A compositional trace-based semantics for probabilistic automata. In *6th Int. Conf. on Concurrency Theory, CONCUR'95, LNCS 962*, pages 234–248. Springer, 1995.

[Seg96] R. Segala. Testing probabilistic automata. In *7th Int. Conf. on Concurrency Theory, CONCUR'96, LNCS 1119*, pages 299–314. Springer, 1996.

[Seg97] R. Segala. Quiescence, fairness, testing, and the notion of implementation. *Information and Computation*, 138(2):194–210, 1997.

[SL95] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[Tre08] J. Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing, LNCS 4949*, pages 1–38. Springer, 2008.

[UW03] H. Ural and D. Whittier. Distributed testing without encountering controllability and observability problems. *Information Processing Letters*, 88(3):133–141, 2003.

[WSS97] S.-H. Wu, S.A. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1-2):1–37, 1997.

[YL92] W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *12th IFIP/WG6.1 Int. Symposium on Protocol Specification, Testing and Verification, PSTV'92*, pages 47–61. North Holland, 1992.