2009 Seventh IEEE International Conference on Software Engineering and Formal Methods

Simulation relations for systems with distributed interfaces*

Robert M. Hierons
Department of Information Systems and Computing
Brunel University, Uxbridge, Middlesex, UB8 3PH United Kingdom
rob.hierons@brunel.ac.uk

Manuel Núñez
Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, Spain
mn@sip.ucm.es

Abstract

In this paper we define simulation relations for distributed systems. Taking as starting point our previous work on the distributed testing architecture, we introduce novel simulation relations that can be used to define, given a specification, what a good implementation is. We approach the problem from two different perspectives. First, we consider that different ports of the system cannot share information. Thus, the decision to consider whether a system is correct has to be based only on local observations. We give some examples to show that this relation is very weak and propose a new one where we allow the different ports to partially communicate. Specifically, we do not implement a complex synchronization mechanism but allow entities to combine whole traces to obtain a verdict.

1 Introduction

The high complexity of current software systems has enforced the use of systematic techniques in order to assess their correctness. One of the software engineering methodologies to perform this assessment consists in working with an abstract model (specification) showing the desirable behaviour of the system. Then, the correctness of the system is defined in terms of its comparison with the specification: We say that the system *conforms* to the specification if it does not show a behaviour that contradicts the model. In order to check the conformance of the developed system with respect to the specification, specially if we have a state based model, the area of *formal testing* has received much attention (see, for example, [LY96, Tre96, Pet01, BT01, HU02, Hie02, PY05, RMN08, HBH08, HBB+09]).

Even though most work on formal testing is based on notions related to trace containment (for its extended use, we can mention **ioco** [Tre96, Tre08]), other possibilities to establish the correctness of the system can be introduced. In this line, simulation relations [Gla93, LV95, Gla01, CFG08, FG09] are a good candidate since they relate two processes if one of them is able to simulate the behaviour of the other. The asymmetry of the notion is more suitable to define what a good implementation is, in contrast with symmetric notions such as bisimulation, because it recognises that the related objects can be, in fact, of different nature. However, this advantage of simulation relations has, to the best of our knowledge, not been exploited before in the context of formal conformance relations for distributed systems.

If the system that we are studying has physically distributed interfaces, called *ports* in this paper, then in order to determine its conformance with respect to a specification, we place an observer at each port. Usually, we have to assume that either the observers cannot communicate with each other or that this communication is costly since it requires the installation of an external network to channel the huge amount of exchanged messages. In addition, we assume that the different observers do not have access to a global clock. In the framework of formal testing it has been established that the use of such a *decentralised* approach reduces the ability to distinguish between agents and both conformance and testing in this context have received much attention (see, for example, [SB84, DB85, LDB93, TY98, RC03, UW06, HU08]).

Taking as a first step our previous work [HMN08a, HMN08b] on formal testing in the distributed architecture, the main purpose of this paper is to define *sensible* simulation relations to establish the conformance of a system

*Research partially supported by the Spanish MEC project WEST/FAST (TIN2006-15578-C02-01) and by the UCM-BSCH programme to fund research groups (GR58/08 - group number 910606).

978-0-7695-3870-9/09 $26.00 © 2009 IEEE
DOI 10.1109/SEFM.2009.22

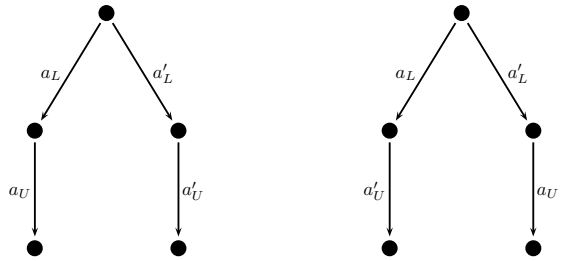

Figure 1. Example of distributed systems.

against a specification in the context of a distributed architecture. In other words, we have to adapt the notion of simulation relation both to a *conformance vision* of the correctness of systems and, more importantly, to deal with the existence of different ports. We consider two scenarios. In the first scenario we assume that ports of the system are independent in the sense that no external agent or system can receive information from more than one of them. In this situation it is sufficient that the local behaviour observed at a port p of the system is consistent with some global behaviour of the specification and that this is the case for every port. In the second scenario there is the possibility that information from two or more of the ports could be received by some external agent/system and as a result the local behaviours observed at the separate ports could be brought together. This leads to a stronger simulation relation where some errors that could not be detected while applying the previous relation can be unveiled.

In order to see the difference, consider a specification s which nondeterministically chooses to either have event a_L at port L followed by a_U at port U or event a'_L at port L followed by a'_U at port U (see Figure 1, left). Further, let us suppose that the implementation r nondeterministically chooses to either have event a_L at port L followed by a'_U at port U or event a'_L at port L followed by a_U at port U (see Figure 1, right). Then r conforms to s under the weaker notion of conformance (simulation) since in each case the agent at port U can either observe a_U or a'_U and the agent at port L can either observe a_L or a'_L. However, r should not conform to s under our stronger notion of conformance since each possible behaviour of the implementation (the performance of a_L and a'_U or the performance of a'_L and a_U) can be distinguished from the behaviours of the specification if we allow an agent to receive information about the local observations made at each port.

The rest of the paper is structured as follows. In Section 2 we introduce some preliminary concepts that we will use along the paper. In particular, we will give a formalism to define systems having distributed ports. In Section 3 we introduce our first notion of distributed simulation that we call *weak simulation*. In Section 4 we point out the drawbacks of the previous simulation relation and introduce an alternative, stronger simulation relation. Let us remark that the use of the adjectives weak and strong is not associated to the usual meaning of these concepts in the context of bisimulation, where the weak notion partially abstract internal actions. In our case both simulation relations are *weak* in this sense since internal actions are dealt with in a way similar to the one in weak bisimulation. In Section 5 we compare our two simulation relations with one another and with the **dioco** relation previously defined [HMN08a, HMN08b]. Finally, in Section 6 we present our conclusions and give some lines for future work.

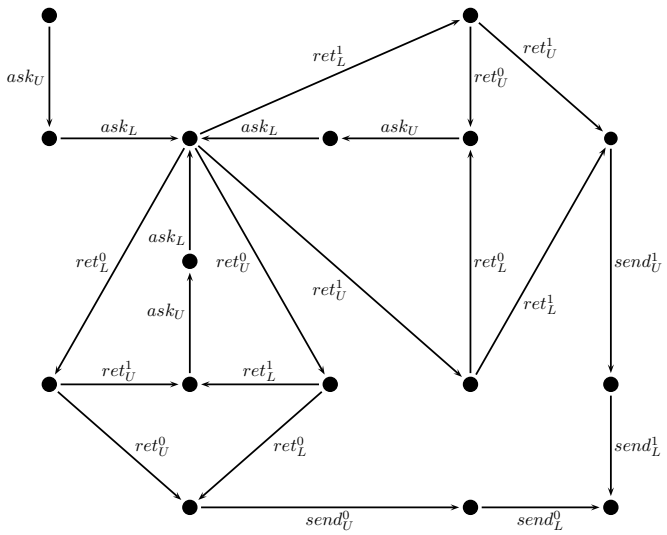

Figure 2. A simple distributed majority voting protocol.

2 Preliminaries

In this section we introduce the main notation that will be used in the paper. In order to present systems we will use a notion of labelled transition system where we take into account the port at which an action is performed. Therefore, we will consider a pairwise disjoint partition of the actions of the system among the different ports. Next, we recall the usual notion of labelled transition system and then show how to consider this type of systems in a context where more than one port is available.

Definition 1 *A labelled transition system s, in short LTS, is defined by the 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 the set of visible actions, and $T \subseteq Q \times (\mathcal{A}ct \cup \{\tau\}) \times Q$, where τ represents internal (unobservable) actions, is the transition*

relation. A transition (q, a, q') means that from state q it is possible to move to state q' with action $a \in \mathcal{A}ct \cup \{\tau\}$.

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

Any state $q \in Q$ induces an LTS derived from s by setting the initial state to q, that is, abusing the notation we consider $q = (Q, \mathcal{A}ct, T, q)$.

During the rest of the paper we use the following notation.

1. *If $(q, a, q') \in T$, for $a \in \mathcal{A}ct \cup \{\tau\}$, then we write $q \xrightarrow{a} q'$.*
2. *We write $q \stackrel{a}{\Longrightarrow} q'$, for $a \in \mathcal{A}ct$, if there exist $q_0, \ldots, q_k$, with $k \geq 1$, and $0 \leq i < k$ such that $q = q_0$, $q' = q_k$, and $q_0 \xrightarrow{\tau} q_1, \ldots, q_{i-1} \xrightarrow{\tau} q_i$, $q_i \xrightarrow{a} q_{i+1}$, $q_{i+1} \xrightarrow{\tau} q_{i+2}, \ldots, q_{k-1} \xrightarrow{\tau} q_k$.*
3. *We write $q \stackrel{\epsilon}{\Longrightarrow} q'$ if there exist $q_0, \ldots, q_k$, with $k \geq 0$, such that $q = q_0$, $q' = q_k$, and $q_0 \xrightarrow{\tau} q_1, \ldots, q_{k-1} \xrightarrow{\tau} q_k$. In particular, we have $q \stackrel{\epsilon}{\Longrightarrow} q$ for all $q \in Q$.*
4. *We write $q \stackrel{\sigma}{\Longrightarrow} q'$ for $\sigma = a_1, \ldots, a_k \in \mathcal{A}ct^*$ if there exist $q_0, \ldots, q_k$, such that $q = q_0$, $q' = q_k$ and for all $1 \leq i \leq k$ we have that $q_{i-1} \stackrel{a_i}{\Longrightarrow} q_i$.*
5. *We write $s \stackrel{\sigma}{\Longrightarrow}$ if there exists q' such that $q_{in} \stackrel{\sigma}{\Longrightarrow} q'$ and we say that σ is a* trace *of s. We let $\mathcal{T}r^*(s)$ denote the set of* traces *of s.*

If a system has multiple interfaces (ports) at which it interacts with its environment then we can adapt our formalism to deal with this situation. Given an LTS $(Q, \mathcal{A}ct, T, q_{in})$ with port set $\mathcal{P} = \{1, \ldots, m\}$, for a port $p \in \mathcal{P}$ we identify the set $\mathcal{A}ct_p$ of visible actions that can be observed at p. This partitions $\mathcal{A}ct$ into $\mathcal{A}ct_1, \ldots, \mathcal{A}ct_m$. We assume that $\mathcal{A}ct_1, \ldots \mathcal{A}ct_m$ are pairwise disjoint.[1] We use the term LTS for the case where there are multiple ports and when there is only one port we use the term single-port LTS.

Example 1 In Figure 2 we sketch, as an LTS, the specification of the *server side* of a simple protocol to perform majority voting. We will use this system as a running example to illustrate some of the concepts that we will introduce in the paper. There are two distributed parties that are asked to vote either 0 or 1 (this is indicated by the two consecutive transitions labelled by ask_U and ask_L). Then, the server receives the answers from each party ret^x_L and ret^x_U. If they voted the same, then the server sends a message to the parties with the corresponding result, that is, either the transitions $send^0_U$ and $send^0_L$ or the transitions $send^1_U$ and $send^1_L$, and if they voted different, then the server resends a request for votes with the previously indicated sequence of transitions ask_U and ask_L.

The corresponding LTS is $s = (Q, \mathcal{A}ct, T, q_{in})$, where Q, the states of the system, is given by the set of different bullets, q_{in}, the initial state, is the state in the top-left corner, $\mathcal{A}ct = \{ask_U, ask_L, ret^0_L, ret^1_L, ret^0_U, ret^1_U, send^0_U, send^0_L, send^1_U, send^1_L\}$ is the set of visible actions, and T, the set of transitions, is given by the directed arcs of the graph.

If we see the previous LTS as a multi-port LTS we have two different ports, that is, $\mathcal{P} = \{L, U\}$, so that $\mathcal{A}ct_U = \{ask_U, ret^0_U, ret^1_U, send^0_U, send^1_U\}$ and $\mathcal{A}ct_L = \{ask_L, ret^0_L, ret^1_L, send^0_L, send^1_L\}$. □

The next definition is based on the corresponding one from our earlier work [HMN08a] but taking into account the peculiarities of the current framework where we do not distinguish between inputs and outputs.

Definition 2 *Let $s = (Q, \mathcal{A}ct, T, q_{in})$ be an LTS with port set $\mathcal{P} = \{1, \ldots, m\}$. Let $p \in \mathcal{P}$ and $\sigma \in \mathcal{T}r^*(s)$ be a trace of s. We let $\pi_p(\sigma)$ denote the projection of σ onto p and this is called a* local trace*. This is formally defined by the following rules:*

1. *$\pi_p(\epsilon) = \epsilon$.*
2. *If $a \in \mathcal{A}ct_p$ then $\pi_p(a\sigma) = a\pi_p(\sigma)$.*
3. *If $a \notin \mathcal{A}ct_p$ then $\pi_p(a\sigma) = \pi_p(\sigma)$.*

Given $\sigma, \sigma' \in \mathcal{T}r^*(s)$ we write $\sigma \sim \sigma'$ if σ and σ' cannot be distinguished when making local observations, that is, for all $p \in \mathcal{P}$ we have that $\pi_p(\sigma) = \pi_p(\sigma')$.

Let $p \in \mathcal{P}$. We denote by $s\rceil_p$ the process $(Q, \mathcal{A}ct_p, T', q_{in})$ formed by replacing in s all actions that do not occur at p by τ. Thus, T' is the minimum set such that for all transition $q \xrightarrow{a} q' \in T$ we have that:

1. *If $a \in \mathcal{A}ct_p \cup \{\tau\}$ then $q \xrightarrow{a} q' \in T'$.*
2. *If $a \in \mathcal{A}ct_q$, for some $q \neq p$, then $q \xrightarrow{\tau} q' \in T'$.*

An alternative approach to define projections is to consider the graph induced by the $\stackrel{a}{\Longrightarrow}$ relation. In this case, we can construct the τ-*less* projected LTS by using the classical $\mathcal{O}(n^{2.376})$ algorithm based on the transitive closure operation of [CW90] that has been previously used to decide weak bisimulation.

Example 2 In Figure 3 we show the projection on port L of our running example. Let us remark that since the information concerning port U has been removed we now have

[1] If the same types of values can be received or sent by the SUT at different ports, then we can label these in order to ensure that the sets are disjoint.

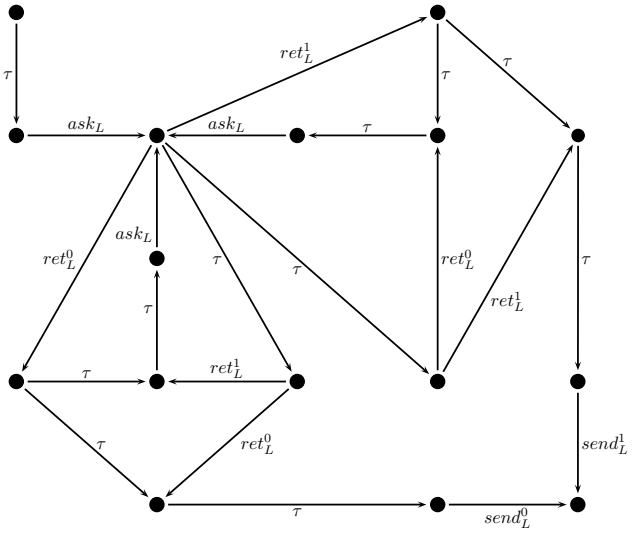

Figure 3. Projection on port L of the majority voting protocol.

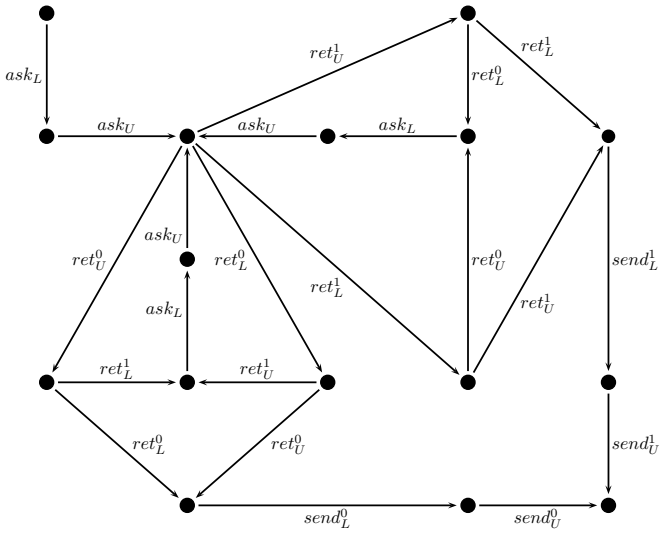

Figure 4. A variant of the simple distributed majority voting protocol.

a *strange* protocol. For example, after producing the action ret_L^0 it will non-deterministically decide either to produce $send_L^0$ at port L, concluding the process, or to produce ask_L at port L, reinitiating the process. Since we cannot see what was produced at port U, the decision is, apparently, non-deterministic. In conclusion, by considering projections, we are losing a lot of information about the causality relations in our protocol, but projections are the right tool to characterize one of our simulation relations. □

3 A weak simulation relation

This paper considers simulation relations for the case when only local observations are made. To see that this differs from normal notions of simulation consider, for example, the majority voting protocol in Figure 2. The operation of this protocol can be seen as one in which there is a sequence of pairs of events, each pair containing exactly one event at each port. As a result, if we only make local observations then we cannot distinguish between this protocol and the one given in Figure 4 since this only differs in the order in which the events in each pair occur.

In one possible scenario, there is an agent at each port of the implementation and no agent can receive information from more than one of these agents. In this situation, we can consider the ports separately: It is sufficient that for every port $p \in \mathcal{P}$ the observations that can be made of the implementation at p are consistent with the observations that can be made of the specification at p. In this section we define a simulation relation for this scenario.

When considering a port p, all events at ports other than p are unobservable and so can be treated in a similar manner to silent moves. This observation leads us to define the following notation where a special notion of *observable* trace is considered: We only observe actions performed at port p.

Definition 3 *Let $s = (Q, \mathcal{A}ct, T, q_{in})$ be an LTS and $p \in \mathcal{P}$ be a port of s. We use the following notation.*

1. *If $(q, a, q') \in T$, for some $a \in \mathcal{A}ct_p$, then we write $q \xrightarrow{a}_p q'$.*
2. *We write $q \stackrel{a}{\Longrightarrow}_p q'$, for $a \in \mathcal{A}ct_p$, if there exist $q_0, \ldots, q_k$, with $k \geq 1$, and $0 \leq i < k$ such that $q = q_0$, $q' = q_k$, $q_0 \xrightarrow{a_1} q_1, \ldots q_{i-1} \xrightarrow{a_i} q_i$, $q_i \xrightarrow{a} q_{i+1}$, $q_{i+1} \xrightarrow{a_{i+1}} q_{i+2}, \ldots, q_{k-1} \xrightarrow{a_{k-1}} q_k$ for $a_1, \ldots, a_{k-1} \in \mathcal{A}ct \cup \{\tau\} \setminus \mathcal{A}ct_p$.*
3. *We write $q \stackrel{\epsilon}{\Longrightarrow}_p q'$ if there exist $q_0, \ldots, q_k$, for $k \geq 0$, such that $q = q_0$, $q' = q_k$, $q_0 \xrightarrow{a_1} q_1, \ldots, q_{k-1} \xrightarrow{a_k} q_k$ for $a_1, \ldots, a_k \in \mathcal{A}ct \cup \{\tau\} \setminus \mathcal{A}ct_p$.*
4. *We write $q \stackrel{\sigma}{\Longrightarrow}_p q'$ for $\sigma = a_1, \ldots, a_k \in \mathcal{A}ct_p^*$ if there exist $q_0, \ldots, q_k$, such that $q = q_0$, $q' = q_k$ and for all $1 \leq i \leq k$ we have that $q_{i-1} \stackrel{a_i}{\Longrightarrow}_p q_i$.*
5. *We write $s \stackrel{\sigma}{\Longrightarrow}_p$ if there exists state $q' \in Q$ such that $q_{in} \stackrel{\sigma}{\Longrightarrow}_p q'$ and we say that σ is a p-*trace *of s. We let $\mathcal{T}r_p^*(s)$ denote the set of p-*traces *of s.*

We can use this notation in order to describe the sequences of observations that can be made at a port. Using

this we can define our first simulation relation. During the rest of the paper we will consider that every time that we relate implementations and specifications they have the same port set.

Definition 4 *Let $s = (Q^s, \mathcal{A}ct, T^s, q^s_{in})$ be a specification and $r = (Q^r, \mathcal{A}ct, T^r, q^r_{in})$ be an implementation with the same port set $\mathcal{P}$. Given $p \in \mathcal{P}$, $\preceq'_p$ is a weak simulation for s and r if for all $a \in \mathcal{A}ct_p$, $q_s \in Q^s$, and $q_r, q'_r \in Q^r$ such that $q_s \preceq' q_r$ and $q_r \stackrel{a}{\Longrightarrow}_p q'_r$ we have that there exists $q'_s \in Q^s$ such that $q_s \stackrel{a}{\Longrightarrow}_p q'_s$ and $q'_s \preceq'_p q'_r$. If there is such a weak simulation $\preceq'_p$ and $q_s \preceq'_p q_r$ then we write $q_s \preceq_p q_r$ and we say that q_s* weakly simulates *q_r at p. So $\preceq_p$ is the largest simulation relation for port p and processes s and r.*

If for all $p \in \mathcal{P}$ we have that q_s weakly simulates q_r at p then we say that q_s weakly simulates *q_r and we write $q_s \preceq q_r$.*

Finally, we write $s \preceq r$ if $q^s_{in} \preceq q^r_{in}$. In this case we say that s weakly simulates *r.*

We can now show that our first simulation relation $\preceq$ has some desirable properties. The proof of the following result is straightforward.

Proposition 1 Let s, r be LTSs with the same port set $\mathcal{P}$. Then, the following properties hold:

- Let $p \in \mathcal{P}$ be a port. If $s \preceq_p r$ then for every trace $\sigma \in \mathcal{T}r^*(r)$ there is some $\sigma' \in \mathcal{T}r^*(s)$ such that $\pi_p(\sigma') = \pi_p(\sigma)$.
- We have that $s \preceq r$ if and only if for every $p \in \mathcal{P}$ we have that $s\rceil_p$ simulates $r\rceil_p$.

□

The following relation allows us to provide an alternative characterisation of our simulation relation $\preceq$. Intuitively, the R relation introduced in the following definition relates two processes if the performance of one action by the implementation at a certain port can be appropriately matched by the specification.

Definition 5 *Let $s = (Q^s, \mathcal{A}ct, T^s, q^s_{in})$ be a specification and $r = (Q^r, \mathcal{A}ct, T^r, q^r_{in})$ be an implementation with the same port set $\mathcal{P}$. We write sR_pr if and only if sR'_pr for a relation R'_p such that for all $q_s \in Q^s$ and $q_r \in Q^r$, if $q_sR'_pq_r$ and $q_r \stackrel{a}{\Longrightarrow}_p q'_r$ for $a \in \mathcal{A}ct_p$ then there exist states q'_s, q''_s, q'''_s and sequences $\sigma, \sigma' \in (\mathcal{A}ct \setminus \mathcal{A}ct_p)^*$ such that $q_s \stackrel{\sigma}{\Longrightarrow} q'_s$, $q'_s \stackrel{a}{\longrightarrow} q''_s$, $q''_s \stackrel{\sigma'}{\Longrightarrow} q'''_s$, and $q'''_s R'_p q'_r$.*

We write sRr if and only if for all $p \in \mathcal{P}$ we have that sR_pr.

Proposition 2 Let $s = (Q^s, \mathcal{A}ct, T^s, q^s_{in})$ be a specification and $r = (Q^r, \mathcal{A}ct, T^r, q^r_{in})$ be an implementation with the same port set $\mathcal{P}$. Then, for $p \in \mathcal{P}$ we have that $s \preceq_p r$ if and only if sR_pr.

Proof : First, let us assume that $s \preceq_p r$. It is sufficient to prove that the choice $R'_p = \preceq_p$ satisfies the requirements of Definition 5. We therefore assume that $q_s \preceq_p q_r$ we have that $q_r \stackrel{a}{\Longrightarrow}_p q'_r$ for some $a \in \mathcal{A}ct_p$ and are required to prove that there exist states q'_s, q''_s, q'''_s and sequences $\sigma, \sigma' \in (\mathcal{A}ct \setminus \mathcal{A}ct_p)^*$ such that $q_s \stackrel{\sigma}{\Longrightarrow} q'_s$, $q'_s \stackrel{a}{\longrightarrow} q''_s$, $q''_s \stackrel{\sigma'}{\Longrightarrow} q'''_s$, and $q'''_s \preceq_p q'_r$. Since $q_s \preceq_p q_r$ there must exist $q'''_s \in Q^s$ such that $q_s \stackrel{a}{\Longrightarrow}_p q'''_s$ and $q'''_s \preceq_p q'_r$. But, by the definition of $q_s \stackrel{a}{\Longrightarrow}_p q'''_s$ there exist $q_1, \ldots, q_{k-1}$, with $k \geq 1$, and $0 < i < k$ such that $q_s \stackrel{a_1}{\longrightarrow} q_1, \ldots q_{i-1} \stackrel{a_i}{\longrightarrow} q_i$, $q_i \stackrel{a}{\longrightarrow} q_{i+1}$, $q_{i+1} \stackrel{a_{i+1}}{\longrightarrow} q_{i+2}, \ldots, q_{k-1} \stackrel{a_{k-1}}{\longrightarrow} q'''_s$ for $a_1, \ldots, a_{k-1} \in \mathcal{A}ct \cup \{\tau\} \setminus \mathcal{A}ct_p$. Therefore, if we consider $q'_s = q_i$, $q''_s = q_{i+1}$, $\sigma = a_1, \ldots, a_i$, and $\sigma' = a_{i+1}, \ldots, a_{k-1}$, then we have that $\sigma, \sigma' \in (\mathcal{A}ct \setminus \mathcal{A}ct_p)^*$ and $q_s \stackrel{\sigma}{\Longrightarrow} q'_s$, $q'_s \stackrel{a}{\longrightarrow} q''_s$, and $q''_s \stackrel{\sigma'}{\Longrightarrow} q'''_s$. Thus, $R'_p = \preceq_p$ satisfies the requirements of Definition 5 and so sR_pr holds as required.

Now let us assume that sR_pr and let R'_p be the largest relation such that if $q_sR'_pq_r$ then for $a \in \mathcal{A}ct_p$ we have that $q_r \stackrel{a}{\Longrightarrow}_p q'_r$ implies that there exist states q'_s, q''_s, q'''_s and sequences $\sigma, \sigma' \in (\mathcal{A}ct \setminus \mathcal{A}ct_p)^*$ such that $q_s \stackrel{\sigma}{\Longrightarrow} q'_s$, $q'_s \stackrel{a}{\longrightarrow} q''_s$, $q''_s \stackrel{\sigma'}{\Longrightarrow} q'''_s$, and $q'''_s R'_p q'_r$. We let $\preceq'_p = R'_p$ and prove that this is a weak simulation. It is sufficient to prove that if $q_s \preceq'_p q_r$ and there exist $a \in \mathcal{A}ct_p$ and $q'_r \in Q^r$ such that $q_r \stackrel{a}{\Longrightarrow}_p q'_r$ then there exists $q'''_s \in Q^s$ such that $q_s \stackrel{a}{\Longrightarrow}_p q'''_s$ and $q'''_s \preceq'_p q'_r$. Since $q_sR'_pq_r$ there exist states q'_s, q''_s, q'''_s and sequences $\sigma, \sigma' \in (\mathcal{A}ct \setminus \mathcal{A}ct_p)^*$ such that $q_s \stackrel{\sigma}{\Longrightarrow} q'_s$, $q'_s \stackrel{a}{\longrightarrow} q''_s$, $q''_s \stackrel{\sigma'}{\Longrightarrow} q'''_s$, and $q'''_s R'_p q'_r$. Thus, since $\preceq'_p = R'_p$ we have that $q'''_s \preceq'_p q'_r$. Finally, by the definition of $\stackrel{a}{\Longrightarrow}_p$ we have that $q_s \stackrel{a}{\Longrightarrow}_p q'''_s$ as required. □

As an immediate corollary of the previous result we obtain that R and $\preceq$ relate the same processes.

Corollary 1 Let $s = (Q^s, \mathcal{A}ct, T^s, q^s_{in})$ be a specification and $r = (Q^r, \mathcal{A}ct, T^r, q^r_{in})$ be an implementation with the same port set $\mathcal{P}$. Then, we have that $s \preceq r$ if and only if sRr.

4 A stronger simulation relation

The simulation relation $\preceq$ corresponds to the situation in which no agent can ever receive information from more than one port. As a result, it is sufficient for the sequence of observations at a port to be consistent with the specification.

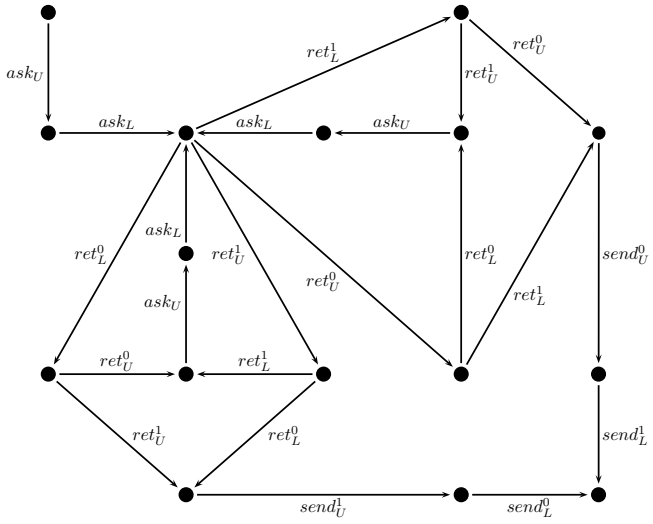

Figure 5. An incorrect simple distributed majority voting protocol.

If, instead, an agent might receive information from more than one port then we need a stronger simulation relation and such a simulation relation is defined in this section.

Example 3 Let us consider the faulty version of the majority voting protocol given in Figure 5. Here we have simply changed the value associated with each reply and send message that occurs at U: from 0 to 1 and from 1 to 0. As a result, for example, if after ask_U and ask_L the events ret_L^1 and ret_U^0 occur then the protocol will send messages $send_U^0$ and $send_L^1$ and then terminate. This is not a correct majority voting protocol since now if L votes 1 and U votes 0, the protocol reports to L that both have voted 1 and reports to U that both have voted 0. However, this looks acceptable to each individual agent since each sees a $send$ message with the same value as the ret message. In fact, the projections of this incorrect protocol at U and L are isomorphic to the projections of the original protocol in Figure 2 at U and L, respectively, and so these two protocols cannot be distinguished under $\preceq$. □

We might simply extend $\preceq$ to consider global traces. However, this is not suitable since if the specification and implementation have produced different traces at some point, future events may lead to traces that are indistinguishable under $\sim$. For example, we would not want to distinguish between a_1a_2 in the specification and a_2a_1 in the implementation if events a_1 and a_2 occur at different ports. In order to define a simulation relation, we add to the state of the specification sequences for each port: These sequences denote additional actions in the trace of the implementation that have not occurred yet in the specification but that might later be 'compensated for' in the specification. If we reach a point where a difference cannot be masked then we know that the implementation cannot be simulated by the specification.

Continuing with the intuitive explanation of our alternative simulation relation, for implementation r and specification s we compare r with $(s, \bar{\epsilon})$, where $\bar{\epsilon}$ denotes the vector of $|\mathcal{P}|$ empty queues. This denotes the processes r and s being in their initial states and so no actions having occurred in either the implementation or the specification. If an action $a \in \mathcal{A}ct_p$ occurs in port p of s and a is the head of the p-queue of s then we remove it from the queue. If a occurs in r at p then we add it to the p-queue for s. If r and s move to states r' and s', respectively, and the queues are empty then the same sequences of observations must have been made at each port. We can represent the possible changes in state and queue contents as defined below.

In defining a simulation relation we will want the specification to delay in simulating an action of the implementation. However, we will also require that if an action a previously performed by the implementation is at the front of the queue of the specification and the specification can currently take a transition with action a then it will do so. We allow the specification to delay simulating an action but do not allow the implementation to delay an action and this is why we only require a queue for s and not for r. This leads us to introduce a notion of *valid change*.

Definition 6 *Let $s = (Q, \mathcal{A}ct, T, q_{in})$ be an LTS with port set $\mathcal{P} = \{1, \ldots, m\}$, for all $p \in \mathcal{P}$ let sq_i be a sequence of visible actions belonging to $\mathcal{A}ct_p$ and let $\overline{sq} = (sq_1, \ldots, sq_m)$. For all $q \in Q$ we say that the pair $(q, \overline{sq})$ is a configuration of s.*

Let $s = (Q^s, \mathcal{A}ct, T^s, q^s_{in})$ be a specification and $r = (Q^r, \mathcal{A}ct, T^r, q^r_{in})$ be an implementation with the same port set $\mathcal{P}$. Given a configuration $(q_s, \overline{sq})$ and a state $q_r \in Q^r$, we say that the following are valid changes*:*

1. *If $q_s \stackrel{a}{\Longrightarrow} q'_s$, for $a \in Act_p$, and $sq_p = asq'_p$ for some sq'_p, then we say that $((q_s, \overline{sq}), q_r) \hookrightarrow_{s,a} ((q'_s, (sq_1, \ldots, sq_{p-1}, sq'_p, sq_{p+1}, \ldots, sq_m)), q_r)$ is a valid change and we also write $((q_s, \overline{sq}), q_r) \hookrightarrow^v ((q'_s, (sq_1, \ldots, sq_{p-1}, sq'_p, sq_{p+1}, \ldots, sq_m)), q_r)$.*

2. *If the premises of the previous item do not hold and $q_r \stackrel{a}{\Longrightarrow} q'_r$, for $a \in Act_p$, then $((q_s, \overline{sq}), q_r) \hookrightarrow_{r,a} ((q_s, (sq_1, \ldots, sq_{p-1}, sq_pa, sq_{p+1}, \ldots, sq_m)), q'_r)$ is a valid change and we also write $((q_s, \overline{sq}), q_r) \hookrightarrow^v ((q_s, (sq_1, \ldots, sq_{p-1}, sq_pa, sq_{p+1}, \ldots, sq_m)), q_r)$.*

Example 4 Let us consider the specification s given in Figure 2 and the implementation r given in Figure 4. Here we will label the states in a manner that is consistent; since both

processes are deterministic the state reached by a sequence of events is uniquely defined.

We compare $(q^s_{in}, \bar{\epsilon})$ and q^r_{in}. Consider now the action ask_L that can occur from the initial state of r but not from the initial state of s. Thus, under this action we can apply rule 2 of Definition 6 and move to the situation in which r is in a new state q'_r, s is still in its initial state, and the queues for s are (ϵ, ask_L). We can now take the action ask_U in r moving to state q''_r and s moves to configuration $(q^s_{in}, (ask_U, ask_L))$. We are now in the situation in which there is an action ask_U at the front of a queue of s such that there is a transition from the current state of s with action ask_U and so we apply Rule 1 of Definition 6. Under this, we change the state of s and remove ask_U from the relevant queue and so r stays in the same state and s moves to configuration $(q'_s, (\epsilon, ask_L))$. We now apply Rule 1 again, and in this case r stays in the state q''_r and s moves to configuration $(q''_s, \bar{\epsilon})$. It is not hard to see that we can continue this process to show that transitions of one process can be always appropriately *compensated* by transitions of the other one. □

We can now define a new simulation relation in terms of reachability. The basic idea is that for any legal choice of moves from the implementation under $\hookrightarrow_r$ there must be some corresponding sequence of moves from the specification under $\hookrightarrow_s$ such that some final state does not represent failure. Failure can occur through it being impossible to simulate some of the actions of r and this can be seen as one or more of the queues associated with s being non-empty.

Definition 7 *Let $s = (Q^s, \mathcal{A}ct, T^s, q^s_{in})$ be a specification and $r = (Q^r, \mathcal{A}ct, T^r, q^r_{in})$ be an implementation with the same port set $\mathcal{P} = \{1, \ldots, m\}$. We say that $\sqsubseteq'$ is a strong simulation if for every configuration $(q_s, \overline{sq})$ of s and state q_r of r, if $(q_s, \overline{sq}) \sqsubseteq' q_r$ then the following hold:*

1. *There is a sequence of valid changes that moves to a situation in which the queue in the configuration for s is empty and that passes through pairs of configurations related under $\sqsubseteq'$. More formally, there exist $(q_{s_1}, \overline{sq}_1), \ldots, (q_{s_k}, \overline{sq}_k)$ and $q_{r_1}, \ldots, q_{r_k}$ such that $(q_{s_1}, \overline{sq}_1) = (q_s, \overline{sq})$, $q_{r_1} = q_r$, for all $1 \leq i < k$ we have that $((q_{s_i}, \overline{sq}_i), q_{r_i}) \hookrightarrow^v ((q_{s_{i+1}}, \overline{sq}_{i+1}), q_{r_{i+1}})$ and $(q_{s_{i+1}}, \overline{sq}_{i+1}) \sqsubseteq' q_{r_{i+1}}$, and $\overline{sq}_k = \overline{\epsilon}$. This says that we must be able to move to a situation in which all actions of the implementation have been simulated in the specification and also where all configurations/state pairs we pass through are related under $\sqsubseteq'$.*
2. *If $((q_s, \overline{sq}), q_r) \hookrightarrow^v ((q'_s, \overline{sq}'), q'_r)$ then $(q'_s, \overline{sq}') \sqsubseteq' q'_r$. This says that if we make a valid change then the new configuration for s should be able to simulate the new state of r.*

We say that $(q_s, \overline{sq})$ strongly simulates *q_r if there is a strong simulation relation $\sqsubseteq'$ such that $(q_s, \overline{sq}) \sqsubseteq' q_r$ and we write $(q_s, \overline{sq}) \sqsubseteq q_r$. We say that s* strongly simulates *r if $(q^s_{in}, \bar{\epsilon})$ strongly simulates q^r_{in} and we write $s \sqsubseteq r$.*

Let us remark that this notion of strong simulation is asymmetric since it requires that the specification should be able to simulate behaviours of the implementation but does not require the implementation to be able to simulate behaviours of the specification.

5 Comparing relations

In this section we compare $\preceq$ and $\sqsubseteq$ with one another and with the previously defined **dioco** relation [HMN08a, HMN08b]. The **dioco** relation represents a conservative extension of the **ioco** relation [Tre96] to the distributed setting. The **dioco** relation operates in a similar manner to **ioco** by comparing suspension traces of the implementation and the specification, where a suspension trace is a trace in which δ can be added whenever a process is in a quiescent state: A state in which it cannot progress without further input.

The original definition of **dioco** [HMN08a, HMN08b] only allowed global traces to be compared in quiescent states. Since we do not distinguish between input and output, we only have quiescence when in a deadlock state, where a state q is a deadlock state if there are no transitions of the form (q, a, q'). In order to adapt **dioco** to the current framework we consider only traces reaching a *deadlocked* state. Thus, we introduce the notion of *complete* trace.

Definition 8 *Let s be an LTS and σ be a finite trace of s. We say that σ is a* deadlocking trace *of s if there exists $q \in Q$ such that $q_{in} \stackrel{\sigma}{\Longrightarrow} q$ and for all $a \in \mathcal{A}ct \cup \{\tau\}$ we have that there does not exist $q' \in Q$ such that $(q, a, q') \in T$. In this case we also say that q is a* deadlocked state *of s. We define the set of* complete traces *of s, denoted by $\mathcal{CT}r(s)$, as the set containing all the deadlocking traces of s.*

Let s, r be LTSs with the same port set $\mathcal{P}$. We write r **dioco** *s if and only if for all $\sigma \in \mathcal{CT}r(r)$ there exists some $\sigma' \in \mathcal{T}r^*(s)$ such that $\sigma \sim \sigma'$.*

Proposition 3 There exist processes r and s such that r **dioco** s but s does not strongly simulate r.

Proof : It is sufficient to consider the processes shown in Figure 6. These are indistinguishable under **dioco** but under $\sqsubseteq$ it is sufficient to observe that after a_L the first process can perform b_U and b'_U and so it cannot be simulated by the second process since this must either be able to perform b_U only or to be able to perform b'_U only. □

As we might expect, given processes r and s, if s does strongly simulate r then r **dioco** s.

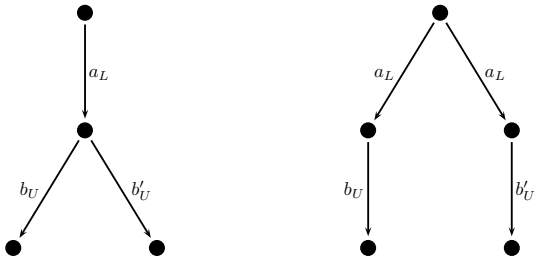

Figure 6. Processes that cannot be distinguished under dioco.

Proposition 4 Let $s = (Q^s, \mathcal{A}ct, T^s, q^s_{in})$ be a specification and $r = (Q^r, \mathcal{A}ct, T^r, q^r_{in})$ be an implementation with the same port set $\mathcal{P}$. If $s \sqsubseteq r$ then r **dioco** s.

Proof: Let us assume that $s \sqsubseteq r$. It is sufficient to prove that for each $\sigma \in \mathcal{CT}r(r)$, there exists some $\sigma' \in \mathcal{T}r^*(s)$ such that $\sigma \sim \sigma'$.

Let q'_r be a deadlocked state of r such that $q^r_{in} \stackrel{\sigma}{\Longrightarrow} q'_r$. Since $s \sqsubseteq r$, there is a sequence ρ of valid moves from $((q^s_{in}, \overline{\epsilon}), q^r_{in})$ to $((q'_s, \overline{s\sigma}), q'_r)$ for some configuration $(q'_s, \overline{s\sigma})$ such that the moves in ρ involving transitions in r have label σ. Since $s \sqsubseteq r$ we must have that there is a sequence ρ' of valid moves from $((q'_s, \overline{s\sigma}), q'_r)$ to some $((q''_s, \overline{\epsilon}), q''_r)$. Since q'_r is a deadlock state we have that $q''_r = q_r$. In addition, s can only simulate actions that have already occurred in r and thus the sequence σ' of actions at s in $\rho\rho'$ must satisfy $\sigma' \sim \sigma$. Thus there exists $\sigma' \in \mathcal{T}r^*(s)$ such that $\sigma' \sim \sigma$ and so the result follows. □

An alternative definition $\sqsubseteq_{alt}$ of strong simulation would have queues for both s and r and allow s to simulate actions of r that have yet to happen and allow the queue for r to be non-empty at the end of a sequence of valid moves. Interestingly, we would not have that $s \sqsubseteq_{alt} r$ not implying that s **dioco** r since while **dioco** allows the specification to do actions in addition to those in a complete trace σ of r, these must occur after a trace σ' such that $\sigma' \sim \sigma$. In contrast, in $\sqsubseteq_{alt}$ we could allow s to take additional actions to produce a trace σ' such that for every port p we have that $\pi_p(\sigma)$ is a prefix of $\pi_p(\sigma')$. This appears to be reasonable, since such a global trace σ' is indistinguishable from another global trace $\sigma'' \sim \sigma'$ such that σ is a prefix of σ''. There thus seems merit in investigating simulation relations similar to $\sqsubseteq_{alt}$ and another interesting challenge is adapting **dioco** in order to make it less restrictive. However, the definition of this new notion goes well beyond the scope of this paper and will be tackled in future work.

We can also compare $\sqsubseteq$ with $\preceq$, obtaining the expected result.

Proposition 5 Let $s = (Q^s, \mathcal{A}ct, T^s, q^s_{in})$ be a specification and $r = (Q^r, \mathcal{A}ct, T^r, q^r_{in})$ be an implementation with the same port set $\mathcal{P}$. We have that if $s \sqsubseteq r$ then $s \preceq r$ but it is possible that $s \preceq r$ but not that $s \sqsubseteq r$.

Proof: First, let us assume that $s \sqsubseteq r$ and let P denote the set of sequences of valid moves from $((q^s_{in}, \overline{\epsilon}), q^r_{in})$. Let us note that we do not restrict P to complete paths and so P is prefix closed. We will define a relation on the basis of the paths in P, parameterized by a port $p \in \mathcal{P}$, and we will show that this relation is a weak simulation for s and r at p.

We write $q_s \mathcal{R}_{p,\rho} q_r$ if $\rho \in P$ is a path such that along ρ the sequence of actions in s at p is $\sigma = a_1, \ldots, a_k$ and the following two conditions hold:

1. q_s is the state of s in the configuration reached by the shortest prefix ρ' of ρ whose sequence of actions in s at p is σ
2. q_r is any state of r on ρ reached by a prefix ρ'' of ρ whose sequence of actions in r at p is σ.

In addition, we write $s\mathcal{R}_p r$ if there exists ρ such that $q^s_{in} \mathcal{R}_{p,\rho} q^r_{in}$. Clearly, we have that $s\mathcal{R}_p r$. Now let us suppose that $q_s \mathcal{R}_p q_r$ and $q_r \stackrel{a}{\Longrightarrow}_p q'_r$. Since $q_s \mathcal{R}_p q_r$ we have that $q_s \mathcal{R}_{p,\rho} q_r$ for some $\rho \in P$ and let the corresponding sequence of actions in s at p be σ. In addition, since $q_r \stackrel{a}{\Longrightarrow}_p q'_r$ and $s \sqsubseteq r$, there is a path in P with prefix ρ whose sequence of actions in s at p is σa and let ρ' denote a shortest such path. Let q'_s denote the state of s in the configuration reached by ρ'. Then by the definition of $\mathcal{R}_p$ we have that $q'_s \mathcal{R}_p q'_r$. In addition, $q_s \stackrel{a}{\Longrightarrow}_p q'_s$. Thus, $\mathcal{R}_p$ is a weak simulation for port p and processes s and r. Since this can be done for any port p we have that s weakly simulates r as required.

To see that it is possible that $s \preceq r$ but not that $s \sqsubseteq r$, let us consider the processes depicted in Figure 1. It is clear that these are related under $\preceq$ since in each case the observation made at port L is either a_L or a'_L and the observation made at port U is either a_U or a'_U. In addition, these two processes are not related by either **dioco** or $\sqsubseteq$ since each complete trace σ of one has the property that no complete trace σ' of the other is equivalent to σ under $\sim$. □

Our definition of $\hookrightarrow$ did not force the implementation and specification to simultaneously use the same action when this is possible. However, our definition of $\sqsubseteq$ did require that the specification should take a transition corresponding to an earlier action of the implementation when this is possible. If we do not make such a restriction, and the implementation has only finite traces, then the specification can wait until the implementation has deadlocked and then simulate the sequence of actions that occurred and as a result we would have a relation that is very similar to trace inclusion.

Now let us consider the processes depicted in Figure 7. Under our definition of $\sqsubseteq$, when one takes a common cur-

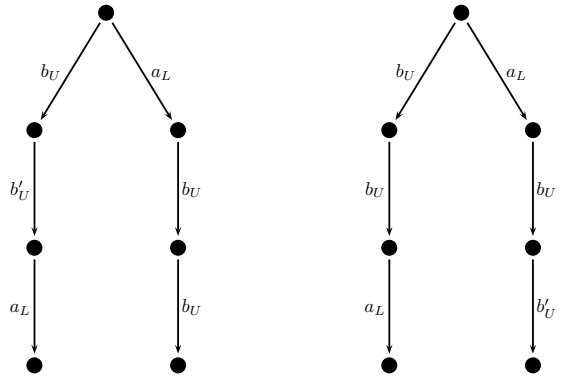

Figure 7. Two equivalent processes.

rent action from the initial state then we have to consider the set of configurations in which the other process takes the same action. As a result we will distinguish between these processes. It could be argued that we should not be able to distinguish between these processes and thus that in some situations $\sqsubseteq$ is too strong. The problem here is that we have branching in which the branches have actions at different ports. Future work will consider alternative simulation relations.

6 Conclusions

Distributed systems have become increasingly important and this has led to interest in the verification of distributed systems and their designs. In this paper we have considered the situation in which a system has physically distributed interfaces, called ports, and observations are made locally at the ports. Recent work has shown that this situation can require conventional notions of conformance, such as **ioco**, to be adapted. However, this is the first paper to propose simulation relations for such systems.

We first considered the situation in which there is a separate agent at each port of the implementation, each agent only observes at its port, and no external agent will receive information from more than one of these agents. In this situation it is sufficient that the observations made at a port are consistent with those in the specification and this led us to define a simulation relation $\preceq$. We then produced an alternative characterisation of $\preceq$.

In some situations an external agent will receive information regarding the observations made at most than one port and then $\preceq$ is too weak. This led us to define a second simulation relation, $\sqsubseteq$, in which we require that the set of observations made at the ports are consistent with the specification. It transpires that $\sqsubseteq$ is stronger than $\preceq$ and is also stronger that the implementation relation **dioco** that has been previously defined.

While $\sqsubseteq$ has many of the desired properties, we gave an example in which it is too strong. This example involved the implementation and specification branching and for the branches involving actions at different ports. The problem is that $\sqsubseteq$ forces the specification to make a choice too early. However, it can be argued that processes should normally branch on actions at the same port. Future work will investigate the problem of defining a simulation relation for situations in which branching can occur on events at different ports.

References

[BT01] E. Brinksma and J. Tretmans. Testing transition systems: An annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 187–195. Springer, 2001.

[CFG08] T. Chen, W. Fokkink, and R. van Glabbeek. Ready to preorder: The case of weak process semantics. *Information Processing Letters*, 109(2):104–111, 2008.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251280, 1990.

[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.

[FG09] D. de Frutos and C. Gregorio. (bi)simulations up-to and canonical preorders for the study of process semantics. *Information and Computation*, 207(2):146–170, 2009.

[Gla93] R. van Glabbeek. The linear time-branching time spectrum II. The semantics of sequential processes with silent moves. In *4th Int. Conf. on Concurrency Theory, CONCUR'93, LNCS 715*, pages 66–81. Springer, 1993.

[Gla01] R. van Glabbeek. The linear time-branching time spectrum I. The semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of process algebra*, chapter 1. North Holland, 2001.

[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.

[Hie02] R.M. Hierons. Comparing test sets and criteria in the presence of test hypotheses and fault domains. *ACM Transactions on Software Engineering and Methodology*, 11(4):427–448, 2002.

[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.

[HU02] R.M. Hierons and H. Ural. Reduced length checking sequences. *IEEE Transactions on Computers*, 51(9):1111–1117, 2002.

[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.

[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.

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

[LY96] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[Pet01] A. Petrenko. Fault model-driven test derivation from finite state models: Annotated bibliography. In *4th Summer School on Modeling and Verification of Parallel Processes, MOVEP'00, LNCS 2067*, pages 196–205. Springer, 2001.

[PY05] A. Petrenko and N. Yevtushenko. Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, 54(9):1154–1165, 2005.

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

[RMN08] I. Rodríguez, M.G. Merayo, and M. Núñez. $\mathcal{HOTL}$: Hypotheses and observations testing logic. *Journal of Logic and Algebraic Programming*, 74(2):57–93, 2008.

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

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software – Concepts and Tools*, 17(3):103–120, 1996.

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

[TY98] K.-C. Tai and Y.-C. Young. Synchronizable test sequences of finite state machines. *Computer Networks and ISDN Systems*, 30(12):1111–1134, 1998.

[UW06] H. Ural and C. Williams. Constructing checking sequences for distributed testing. *Formal Aspects of Computing*, 18(1):84–101, 2006.