Testing Input/Output Partial Order Automata

Stefan Haar[1], Claude Jard[2], and Guy-Vincent Jourdan[3]

[1] IRISA/INRIA
Rennes, France
Stefan.Haar@irisa.fr
[2] IRISA, ENS Cachan Bretagne
Campus de Ker-Lann, F-35170 Bruz, France
Claude.Jard@bretagne.ens-cachan.fr
[3] School of Information Technology and Engineering (SITE)
University of Ottawa
800 King Edward Avenue
Ottawa, Ontario, Canada, K1N 6N5
gvj@site.uottawa.ca

Abstract. We propose an extension of the Finite State Machine framework in distributed systems, using *input/output partial order automata (IOPOA)*. In this model, transitions can be executed non-atomically, reacting to asynchronous inputs on several ports, and producing asynchronous output on those ports. We develop the formal framework for distributed testing in this architecture and compare with the synchronous I/O automaton setting. The advantage of the compact modelling by IOPOA combines with low complexity : the number of tests required for concurrent input in our model is polynomial in the number of inputs.

1 Introduction

Finite State Machines (FSMs) have been used to model many types of sequential systems. However, it is distributed applications over networks that become increasingly important; they do not fit into this sequential model, because inputs may be applied simultaneously and events are not necessarily totally ordered. In the context of testing, distributed testing models use *multi-port* automata in which each transition is guarded by a required vector of inputs (possibly $\perp$, i.e. no input on some channels) on a collection of channels, and produces a vector of outputs (possibly $\perp$) on those channels. This model, often called *Multiports Deterministic FSM* in the literature, but that we call *sequential input automata* in this paper, has been widely studied from a distributed system testing perspective; emphasis is given in that work to the coordination messages, between testers at different ports, that are necessary to avoid *controllability* and *observability* problems in distributed systems testing [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. However, this model is intrinsically sequential regarding the inputs, which must be specified one at a time (although one such single input can generate several, concurrent outputs on different ports). In order to specify that from a given state, two concurrent inputs a and b are required, one has to specify either 'a

A. Petrenko et al. (Eds.): TestCom/FATES 2007, LNCS 4581, pp. 171–185, 2007.

then b' or 'b then a'. Consider the more detailed example in Figure 1. In that context, we need to specify that in order to go from state $\mathbf{s}_i$ to state $\mathbf{s}_f$, we need to input i_1, i_2 and i_3 on ports 1, 2 and 3 respectively. On port 1, the output o_1 should be produced after i_1 was input. On port 3, output o_3 should be produced after i_3 was input, and on port 2, o_2 should be produced after i_1, i_2 and i_3 have all been input. In general, when n inputs must be provided concurrently, the only option is to enumerate all $n!$ ordering for the inputs, leading to a specification that is large and difficult to create, hard to interpret and thus to understand, and whose size makes it difficult to test. Another approach would be to arbitrarily impose a given ordering for the inputs, which seems a poor option and which adds needless constraints at the implementation level.

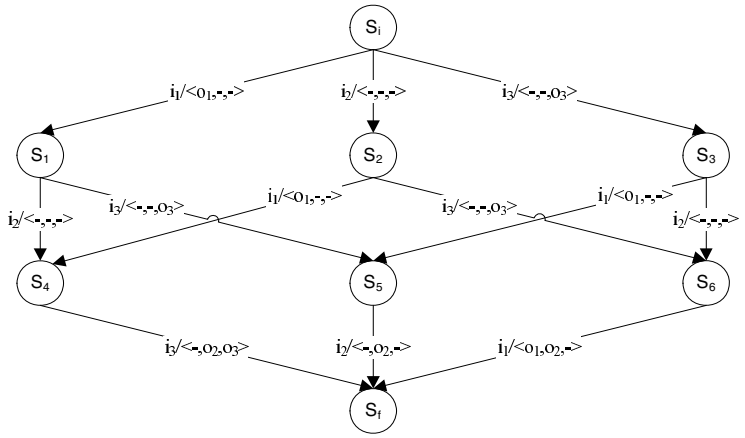

Fig. 1. (Partial) Multiports Deterministic FSM

We therefore endeavour to explore a model that allows specifications to relax synchronization constraints: equipping partial order automata with input/output capabilities. We define a class of *IO-PO-automata (IOPOA)* in which

- inputs can arrive asynchronously, and
- transitions may occur partially, and in several steps, reacting to inputs as they arrive and producing outputs as soon as they are ready, without dedicated synchronization.

The important additional feature (in addition to state transition and output production) of transitions is then a *causal order*: for p channels, we have a bipartite graph of (p inputs) $*$ (p outputs) such that input on channel i precedes output on channel i produced by that transition. Cross-channel dependencies may persist between input on some channel j and output on channel $i \neq j$; at most, the input on j can trigger a broadcast to all channels. However, inputs are not ordered among one another, neither are outputs.

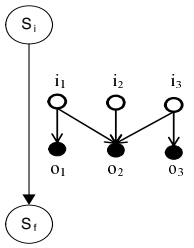

Fig. 2. The IOPOA corresponding to the multiports deterministic FSM of Figure 1

Figure 2 shows the IOPOA corresponding to the set of transitions of Figure 1. Clearly, the result is a much simpler model, with two states and one transition. The role of states for testing in this model will be redefined, and a new role emerges for the partially ordered patterns ; the theoretical toolbox of distinguishing sequences etc. needs to be adapted, yet keeps its importance. Concerning the complexity of checking, one might have expected that the new model were just a concise way of specifying the same behavior, and thus that testing would be the same in both cases (that is, that all combinations of concurrent inputs would have to be tested anyways). It turns out not to be the case; in fact, the number of tests required for concurrent input in our model is *polynomial* in the number of inputs.

The rest of the paper is structured as follows. In section 2, the IOPOA Framework is introduced, section 3 focuses on the differences between the partial order model and the classical one regarding conformance testing. Finally, section 4 discusses future extensions to more general IOPOA classes, and shows that IOPOA can have homing and synchronizing sequences, but may not have state identification or state verification sequences. Section 5 concludes.

2 IOPOA Framework

We introduce the model of IOPO Automaton with I/O vector sequences. The definition of *conformance*, given in 2.3 in this framework needs the notions of well-behavedness and of completion, which we discuss in 2.4.

2.1 IOPO Automata

Definition 1. *An* Input/Output Partial Order Automaton (or IOPO Automaton, IOPOA) *is a tuple* $\mathcal{M} = (S, \mathbf{s^{in}}, Chn, \mathcal{I}, \mathcal{O}, \delta, \lambda, \omega)$*, where*

1. *S is a finite set of* states *and $s_1 = \mathbf{s^{in}} \in S$ is the* initial state*; the number of states of $\mathcal{M}$ is denoted $n \triangleq |S|$ and the states of $\mathcal{M}$ are enumerated, giving $S = \{s_1, \ldots, s_n\}$;*
2. *$Chn = \pi_1, \ldots, \pi_p$ is the set of I/O channels (ports),*

3. *$\mathcal{I}$ is the common input alphabet, and $\mathcal{O}$ the common output alphabet for all channels. Note that the literature often notes different alphabets $\mathcal{I}_1, \ldots, \mathcal{I}_p$ for different channels ; the above implies no loss of generality provided that the port to which an input is applied is uniquely identifiable. Taking*

$$\overline{\mathcal{I}} \triangleq \bigcup_{i=1}^{p} \mathcal{I}_i \; ; \mathcal{I} \triangleq \overline{\mathcal{I}} \times \mathit{Chn},$$

such that (a, i) denotes input a on port i, one can switch from one representation to the other. We require a special symbol $\perp \in \mathcal{I} \cap \mathcal{O}$ to represent empty input/output. Let Θ be the p-tuple $\Theta \triangleq (\perp, \ldots, \perp)$, and

$$\mathcal{X} \triangleq \mathcal{I}^p \backslash \{\Theta\}, \; \mathcal{X}_\Theta \triangleq \mathcal{X} \cup \{\Theta\}$$
$$\mathcal{Y} \triangleq \mathcal{O}^p$$

be the sets of input/output p-vectors, respectively.

4. *$\delta : S \times \mathcal{X} \to S$ is a **(partial) next state function**: $s' = \delta(s, \mathbf{x})$ for states $s, s' \in S$ and $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p) \in \mathcal{X}$ means that if $\mathcal{M}$ is in state s, and inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$ are applied to ports $1, 2, \ldots, p$, respectively, then $\mathcal{M}$ will enter state s';*
5. *$\lambda : S \times \mathcal{X} \to \mathcal{Y}$ is the **output function**; if $\mathcal{M}$ is in state s, and input $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p) \in \mathcal{X}$ is applied, then the output $\lambda(s, \mathbf{x}) = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_p)$ is observed; write $\lambda_i(s, \mathbf{x}) = \mathbf{y}_i$ to indicate that $\mathbf{y}_i$ is observed at port i;*
6. *ω is a **PO transition label function**: For any $(s, \mathbf{x}) \in S \times \mathcal{X}$ such that $\delta(s, \mathbf{x}) = s'$ and $\lambda(s, \mathbf{x}) = \mathbf{y} \in \mathcal{Y}$, $\omega(s, \mathbf{x}) \subseteq (\{\mathbf{x}_1, \ldots, \mathbf{x}_p\} \times \{\mathbf{y}_1, \ldots, \mathbf{y}_p\})$ is a partial order that satisfies*
   (a) $\mathbf{x}_i < \mathbf{y}_i$ for all $i \in \{1, \ldots, p\}$ such that $\mathbf{x}_i \neq \perp$ and $\mathbf{y}_i \neq \perp$, and
   (b) if $\mathbf{x}_i = \perp$, then $\mathbf{x}_i \not\leq \mathbf{y}_j$ for all $j \in \mathit{Chn}$.

We assume throughout this paper that the underlying transition graph is strongly connected for all IOPOA considered. δ and λ extend to sequence-valued functions $S \times \mathcal{X}^* \to S^*$ and $S \times \mathcal{X}^* \to \mathcal{Y}^*$, which we denote by the same function names.

2.2 I/O Vector Sequences

We allow I/O with restricted concurrency. That is, in each round, one input may be given and one output be received on each channel, and I/O on different channels in that round are pairwise concurrent; in particular, inputs can be made in any order. By contrast, I/Os in different rounds are never concurrent: earlier rounds strictly precede all subsequent ones, for all channels.

For $\mathbf{x}, \mathbf{x}' \in \mathcal{X}_\Theta$, say that $\mathbf{x} \leq \mathbf{x}'$ iff for all $i \in \{1, \ldots, p\}$, $\mathbf{x}_i \neq \mathbf{x}'_i$ implies $\mathbf{x}_i = \perp$. Write $\mathbf{x} < \mathbf{x}'$ iff $\mathbf{x} \leq \mathbf{x}'$ and $\mathbf{x} \neq \mathbf{x}'$. Intuitively, if $\mathbf{x} < \mathbf{x}'$, $\mathbf{x}$ can be seen as an *incomplete input* of $\mathbf{x}'$; one may "enter $\mathbf{x}$ first, and later add the rest of $\mathbf{x}'$". This is in fact a key to our technique for transition identification, see below. For vector sequences $\alpha, \beta \in \mathcal{X}^*$, write $\alpha \sqsubseteq \beta$ iff

1. $\alpha_1 \ldots \alpha_{|\alpha|-1}$ is a prefix of β, and
2. $\alpha_{|\alpha|} \leq \beta_{|\alpha|}$.

Note that this is more restrictive than the general partial order prefix relation.

Subtraction:

- For vectors $\mathbf{x} \leq \mathbf{x}'$, let $\mathbf{x}' \ominus \mathbf{x}$ be the vector w such that $w_i = \mathbf{x}'_i$ iff $\mathbf{x}_i = \perp$, and $w_i = \perp$ otherwise.
- For vector sequences $\alpha \sqsubseteq \beta$, let

$$\beta \ominus \alpha \triangleq (\beta_{|\alpha|} \ominus \alpha_{|\alpha|}) \circ \beta_{|\alpha|+1} \ldots \beta_{|\beta|}.$$

2.3 Completion of an IOPOA

Intermediate states: Suppose states s, s' and vectors $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ such that $\delta(s, \mathbf{x}) = s'$ and $\Theta < \mathbf{x}' < \mathbf{x}$. In general, $\delta(s, \mathbf{x}')$ may be undefined; remedy this by using an extended state space, with an intermediate state $s^{\mathbf{x}'} \notin S$ such that input $\mathbf{x}'$ leads from s to $s^{\mathbf{x}'}$, and input $\mathbf{x} \ominus \mathbf{x}'$ leads from $s^{\mathbf{x}'}$ to s'. Formally, we extend S to a superset $\overline{S}$ and assume δ, λ, ω extend to partial functions $\overline{\delta} : (\overline{S} \times \mathcal{X}) \to \overline{S}$, $\overline{\lambda} : (\overline{S} \times \mathcal{X}) \to \mathcal{Y}$ and $\overline{\omega} : (\overline{S} \times \mathcal{X}) \to 2^{(\mathcal{X} \times \mathcal{Y})}$ such that the following properties hold:

1. $\overline{\delta}_{|(S \times \mathcal{X})} \equiv \delta$, $\overline{\lambda}_{|(S \times \mathcal{X})} \equiv \lambda$, and $\overline{\omega}_{|(S \times \mathcal{X})} \equiv \omega$;
2. **Monotonicity:** Changing the order in which inputs are received must not alter the behavior of $\overline{\delta}$, $\overline{\lambda}$ and $\overline{\omega}$. Formally, $\alpha \sqsubseteq \beta$ must imply for all $s \in S$ ($\circ$ denotes concatenation):
   (a) $\overline{\delta}(s, \beta) = \overline{\delta}(\overline{\delta}(s, \alpha), \beta \ominus \alpha)$;
   (b) $\overline{\lambda}(s, \beta) = \overline{\lambda}(s, \alpha) \circ \overline{\lambda}(\overline{\delta}(s, \alpha), \beta \ominus \alpha)$;
   (c) $\overline{\omega}(s, \beta) = \overline{\omega}(s, \alpha) \circ \overline{\lambda}(\overline{\delta}(s, \alpha), \beta \ominus \alpha)$;

If the above are satisfied by $\mathcal{M}$, we say that $\mathcal{M}$ is **well-behaved**. If $\mathcal{M} \triangleq (S, \mathbf{s^{in}}, \mathcal{I}, \mathcal{O}, \mathit{Chn}, \delta, \lambda, \omega)$ is well-behaved, call $\overline{\mathcal{M}} \triangleq (\overline{S}, \mathbf{s^{in}}, \mathcal{I}, \mathcal{O}, \mathit{Chn}, \overline{\delta}, \overline{\lambda}, \overline{\omega})$ its **completion**.

Well-behavedness captures the *strong input determinism* of a transition in a IOPOA. If one transition specifies several inputs, then necessarily these inputs are concurrent, and thus can be input in the system in any order without impact on the state reached at the end of the transition. This is a reasonable assumption since if the state reached was different for different orderings of the input, it would imply that the inputs were in fact causally related, and therefore the specification should not have treated them as concurrent.

Thus, in the following, we require all IOPOAs to be well-behaved, thus we are dealing with strongly deterministic IOPOAs for which no order needs to be enforced for concurrent inputs.

2.4 Morphisms and Conformance

Let $\mathcal{M}$ and $\mathcal{M}'$ be two IOPO automata over the same in/output alphabets:

$$\mathcal{M} = (S, s_1, \mathcal{I}, \mathcal{O}, \mathit{Chn}, \delta, \lambda, \omega)$$
$$\textit{and } \mathcal{M}' = (S', s_2, \mathcal{I}, \mathcal{O}, \mathit{Chn}, \delta', \lambda', \omega').$$

A **morphism** from $\mathcal{M}$ to $\mathcal{M}'$ is a total mapping $\Phi : S \to S'$ with the property that for all $(s, \mathbf{x}) \in S \times \mathcal{X}$ such that $\delta(s, \mathbf{x})$ is defined,

1. $\delta'(\Phi(s), \mathbf{x})$ is defined, and $\delta'(\Phi(s), \mathbf{x}) = \Phi(\delta(s, \mathbf{x}))$;
2. $\lambda'(\Phi(s), \mathbf{x}) = \lambda(s, \mathbf{x})$;
3. Φ induces a partial order isomorphism $\omega(s, \mathbf{x}) \to \omega'(\Phi(s), \mathbf{x})$.

We say that $\mathcal{M}'$ **conforms** to $\mathcal{M}$ iff there exists a *bijective* morphism $\Phi : S \to S'$, called a *conformal map*. Φ is an **isomorphism** iff (i) it is bijective and (ii) Φ^{-1} is a morphism from $\mathcal{M}'$ to $\mathcal{M}$. Note that conformance is not a symmetric relation, and strictly weaker than isomorphism. We note that:

Lemma 1. *The composition of conformal maps yields a conformal map, i.e. conformance is transitive.*

Theorem 1. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be well-behaved IOPO automata. If $\mathcal{M}_2$ conforms to $\mathcal{M}_1$ under $\Phi : S_2 \to S_1$, then $\overline{\mathcal{M}_2}$ conforms to $\overline{\mathcal{M}_1}$.*

Proof. Suppose $\mathcal{M}_2$ conforms to $\mathcal{M}_1$ under $\Phi : S_2 \to S_1$. Let u_1 be an intermediate state of $\overline{\mathcal{M}}$, and $(s_1, \alpha) \in S_1 \times \mathcal{X}^*$ such that $\overline{\delta}_1(s_1, \alpha) = u_1$. By construction of $\overline{\mathcal{M}_1}$, there exists $s_1' \in S_1$ and $\alpha' \in \mathcal{X}^*$ such that $\alpha \sqsubseteq \alpha'$ and

$$\overline{\delta}_1(s_1, \alpha') = \delta_1(s_1, \alpha') = u_1'. \tag{1}$$

Isomorphism of $\mathcal{M}_1$ and $\mathcal{M}_2$ implies that

$$\overline{\delta}_2(s_2, \alpha') = \delta_2(s_2, \alpha') = u_2', \tag{2}$$

where $s_2 \triangleq \Phi(s_1)$, $s_2' \triangleq \Phi(s_1')$, and $u_2' \triangleq \Phi(u_1')$. By construction of $\overline{\mathcal{M}_2}$, there exists an intermediate state u' of $\overline{\mathcal{M}'}$ such that $\overline{\delta}_2(s_2, \alpha) = u_2$. Input determinism implies that u_2 is unique with this property. Set $\overline{\Phi}(u_1) \triangleq u_2$. One obtains an extension $\overline{\Phi} : \overline{S_1} \to \overline{S_2}$ of $\Phi : S_1 \to S_2$, and checks that $\overline{\Phi}$ is bijective and defines a morphism $\overline{\mathcal{M}_1} \to \overline{\mathcal{M}_2}$.

3 Conformance Testing for Automata with Distinguishing Sequences

The utility of the theorem 1 lies in the following application: Suppose we are given an implementation $\mathcal{M} = (S, \mathbf{s^{in}}, \mathit{Chn}, \mathcal{I}, \mathcal{O}, \overline{\delta}, \overline{\lambda}, \overline{\omega})$ and a specification $\mathcal{M}_1 = (S_1, \mathbf{s_1^{in}}, \mathit{Chn}, \mathcal{I}, \mathcal{O}, \delta_1, \lambda, \omega)$. Let $\mathcal{L}_1 \subseteq \mathcal{X}^*$ be the set of all input vector sequences α such that $\delta_1(\mathbf{s_1^{in}}, \alpha)$ is defined, i.e. application of α in $\mathbf{s^{in}}$ takes $\mathcal{M}_1$ to some specification state $s^\alpha = \delta(\mathbf{s_1^{in}}, \alpha) \in S_1$. Let $\mathcal{M}_2$ be the IOPO automaton obtained by applying $\mathcal{L}_1$ in $\mathcal{M}$, i.e. let

$$\begin{aligned} & & \mathcal{M}_2 &\triangleq (S_2, \mathbf{s_1^{in}}, \mathcal{I}, \mathcal{O}, \mathit{Chn}, \delta_2, \lambda_2, \omega_2), \\ & \textit{where}: & S_2 &\triangleq \left\{ s \in S \mid \exists\ \alpha \in \mathcal{L}_1 :\ \delta(\mathbf{s^{in}}, \alpha) = s \right\}, \\ & & \delta_2 &\triangleq \delta_{|S_2 \times \mathcal{L}_1}, \\ & & \lambda_2 &\triangleq \lambda_{|S_2 \times \mathcal{L}_1}, \\ & & \omega_2 &\triangleq \omega_{|S_2 \times \mathcal{L}_1}. \end{aligned}$$

Here, $\overline{\mathcal{L}}_1$ denotes the closure of $\mathcal{L}_1$ under subtraction of prefixes. By construction, $\mathcal{M}$ conforms to $\overline{\mathcal{M}}_2$. Using well-known techniques [17], conformance of $\mathcal{M}_2$ to $\mathcal{M}_1$ can be tested. If the test is passed, we know by Theorem 1 that $\overline{\mathcal{M}}_2$ conforms to $\overline{\mathcal{M}}_1$; thus Lemma 1 yields that $\mathcal{M}$ conforms to $\mathcal{M}_1$. Hence the task of testing conformance for IOPO automata is indeed completed.

In order to actually perform a test of conformance, we use a *checking sequence*. Let $C(\mathcal{M})$ be the set of IOPOA having no more states than $\mathcal{M}$, the same number of ports and the same input and output alphabet.

Definition 2 (Checking Sequence). *Let $\mathcal{M}_1 = (S_1, \mathbf{s}_1^{\mathbf{in}}, Chn, \mathcal{I}, \mathcal{O}, \delta_1, \lambda_1, \omega_1)$ be an IOPOA. A* checking sequence *of $\mathcal{M}_1$ is an input sequence I which distinguishes $\mathcal{M}_1$ from any IOPOA $\mathcal{M}_2 = (S_2, \mathbf{s}_2^{\mathbf{in}}, \mathcal{I}, \mathcal{O}, Chn, \delta_2, \lambda_2, \omega_2)$ in $C(\mathcal{M}_1)$ that does not conform to $\mathcal{M}_1$, i.e. such that $\forall \mathbf{s} \in S_2, \overline{\lambda_1}(\mathbf{s}_1^{\mathbf{in}}, I) \neq \overline{\lambda_2}(\mathbf{s}, I)$ or $\overline{\omega_1}(\mathbf{s}_1^{\mathbf{in}}, I) \neq \overline{\omega_2}(\mathbf{s}, I)$.*

Distinguishing sequences are usually defined as a sequence of inputs that will produce a different output for every state [17]. In the case of IOPOAs, we need to expand this definition to include the possibility of having the same output but different partial order labels.

Definition 3 (Distinguishing Sequence). *An IOPOA $\mathcal{M}$ admits an* adaptive distinguishing sequence *if there is a set of n input sequences $\{\xi_1, \ldots, \xi_n\}$, one per state of S, such that for all $i, j \in [1, \ldots, n], i \neq j$, ξ_i and ξ_j have a non-empty common prefix ξ_{ij} and $\lambda(s_i, \xi_{ij}) \neq \lambda(s_j, \xi_{ij})$ or $\omega(s_i, \xi_{ij}) \neq \omega(s_j, \xi_{ij})$.*

The automaton has a preset distinguishing sequence *if there is an adaptive one such that for all $i, j \in [1, \ldots, n], \xi_i = \xi_j$; in that case, ξ_i distinguishes state s_i.*

Not all automata have adaptive distinguishing sequences, but by definition, if an automaton has a preset checking sequence, it has an adaptive one.

3.1 Assumptions

In the following, we assume that the number q of states in the implementation does not exceed the number of states in the specification, i.e. $q \leq n$. We also assume that the directed graph induced by δ on S in strongly connected (and thus, by construction, the directed graph induced by $\bar{\delta}$ on $\bar{S}$ is also strongly connected). We finally assume that the IOPOA has an adaptive distinguishing sequence.

3.2 Sequential Input Automata

Since sequential input automata form a special case of IOPOA, it is instructive to look at that class first. It is known that we can construct a checking sequences of polynomial length [18, 19, 17], using polynomial time algorithms [20]. One example of such an algorithm is the following [19]. We call a *transfer sequence* $\tau(\mathbf{s}_i, \mathbf{s}_j)$ a sequence taking the machine from state $\mathbf{s}_i$ to state $\mathbf{s}_j$. Such a sequence

always exists, since the state graph is strongly connected. In order to prove the morphism between the specification and the implementation, we need to show that every state on the specification exists in the implementation, and that every transition of the specification is in the implementation as well, going from the correct state to the correct state and generating the correct output when given the correct input.

Assuming that the machine starts in its initial state $\mathbf{s^{in}} = \mathbf{s}_1$ and that we have a distinguishing sequence ξ_i for every state s_i, the following test sequence checks that the implementation has n states, each of which reacts correctly when input the distinguishing sequence for that state:

$$\xi_1 \circ \tau(\delta(\mathbf{s}_1, \xi_1), \mathbf{s}_2) \circ \xi_2 \circ \tau(\delta(\mathbf{s}_2, \xi_2), \mathbf{s}_3) \circ \ldots \circ \xi_n \circ \tau(\delta(\mathbf{s}_n, \xi_n), \mathbf{s}_1) \circ \xi_1 \quad (3)$$

In order to test a transition a/b going from state $\mathbf{s}_i$ to $\mathbf{s}_j$, assuming the implementation is currently in a state $\mathbf{s}_k$, we can use the following test sequence:

$$\tau(\mathbf{s}_k, \mathbf{s}_{i-1}) \circ \xi_{i-1} \circ \tau(\delta(\mathbf{s}_{i-1}, \xi_{i-1}), \mathbf{s}_i) \circ a \circ \xi_j \quad (4)$$

Applying the test sequence 3, then applying the test sequence 4 for each transition provides a checking sequence. Unfortunately, this simple approach will not directly work with IOPOAs, because causal relationships between inputs and outputs between processes are not directly observable. In order to overcome this issue, we need to create longer test sequences that check causal relationships as well.

In order to explain our solution, we first illustrate our technique on a single transition, assuming that the implementation is correct.

3.3 Complete Transition Identification

We will test transitions by *delaying* input on only one channel, i; let us formalize this as input in the **i-test mode**: Let $1 \leq i \leq p$, and suppose an input vector $\mathbf{x} \in \mathcal{X}$ given. Then define input vector $\check{\mathbf{x}}^i$ as

$$\check{\mathbf{x}}^i_j \triangleq \begin{cases} \perp : i = j \\ \mathbf{x}_j : i \neq j, \end{cases}$$

and let $\hat{\mathbf{x}}^i \triangleq \mathbf{x} \ominus \check{\mathbf{x}}^i$; i.e.

$$\hat{\mathbf{x}}^i_j \triangleq \begin{cases} \mathbf{x}_i : i = j \\ \perp : i \neq j. \end{cases}$$

Let $\mathbf{x}$ be an input vector occurring in some input sequence $\alpha = \alpha_1 \ldots$, such that $\alpha_m = \mathbf{x}$ for some m. Applying $\mathbf{x}$ in **i-test mode** in α means applying, instead of α, the sequence $\alpha' \triangleq \alpha_1 \ldots \alpha_{m-1} \check{\mathbf{x}}^i \hat{\mathbf{x}}^i \alpha_{m+1} \ldots$.

Denote as $\Delta_i(\alpha)$ the sequence obtained from $\alpha = \alpha_1 \ldots$ by replacing *each* α_k by the pair $\check{\alpha}^i_k \hat{\alpha}^i_k$, i.e. in $\Delta_i(\alpha)$, input i is delayed in *all* rounds. It is important

to note that delaying creates *equivalent* sequences, in the sense that for all α and i,

$$\begin{aligned}\lambda\left(\Delta_i(\alpha)\right) &= \lambda(\alpha),\\ \delta\left(\Delta_i(\alpha)\right) &= \delta(\alpha),\\ \text{and } \omega\left(\Delta_i(\alpha)\right) &= \omega(\alpha).\end{aligned}$$

Fix some input vector $\mathbf{x}$ and state $\mathbf{s}$, and set $\mathbf{y} \triangleq \lambda(\mathbf{s}, \mathbf{x})$. Assume we are interested in the label $\omega(\mathbf{s}, \mathbf{x})$; more precisely, since input and output are given, look for the partial order $<_\omega \subseteq (\mathbf{x} \times \mathbf{y})$. Denote as $\tau_{\mathbf{s}}^{\mathbf{x}} \triangleq \tau(\delta(\mathbf{s}, \mathbf{x}), \mathbf{s})$ a sequence that brings the machine back to state $\mathbf{s}$ after having input $\mathbf{x}$ from state $\mathbf{s}$. The test is now performed by inputting

$$\sigma \triangleq \check{\mathbf{x}}^1 \hat{\mathbf{x}}^1 \tau_{\mathbf{s}}^{\mathbf{x}} \check{\mathbf{x}}^2 \hat{\mathbf{x}}^2 \tau_{\mathbf{s}}^{\mathbf{x}} \ldots \tau_{\mathbf{s}}^{\mathbf{x}} \check{\mathbf{x}}^p \hat{\mathbf{x}}^p \tau_{\mathbf{s}}^{\mathbf{x}}, \tag{5}$$

that is, return to state $\mathbf{s}$ and test the same input vector $\mathbf{x}$, delaying a different channel in each round. Call $\mathbf{s}_i \triangleq \delta(\check{\mathbf{x}}^i, \mathbf{s})$ and $\check{\mathbf{y}}^i \triangleq \lambda(\check{\mathbf{x}}^i, \mathbf{s})$. Now, exactly those outputs that are generated only after input $\mathbf{x}_i$ are causal consequences of $\mathbf{x}_i$. That is, we obtain $<_\omega$ as follows:

$$<_\omega \triangleq \{(\mathbf{x}_i, \mathbf{y}_i) \mid i \in \mathit{Chn}, \mathbf{x}_i \neq \perp \text{ and } \mathbf{y}_i \neq \perp\} \tag{6}$$

$$\cup \left\{(\mathbf{x}_i, \mathbf{y}_j) \mid j \in \mathit{Chn} - \{i\} \ \wedge \ \check{\mathbf{y}}_j^i = \perp \ \wedge \ \mathbf{y}_j \neq \perp\right\}. \tag{7}$$

In fact, consider $i \neq j$ and $\mathbf{x}_i \neq \perp$ and $\mathbf{y}_j \neq \perp$.

- If $\mathbf{x}_i <_\omega \mathbf{y}_j$, then output $\mathbf{y}_j$ cannot be produced before input $\mathbf{x}_i$ arrives, hence $\check{\mathbf{y}}_j^i = \perp$; and
- conversely, if $\mathbf{x}_i \not<_\omega \mathbf{y}_j$, then $\mathbf{y}_j = \perp$.

Note that we assume here that all enabled outputs are produced and observed immediately, that is, we can actually decide whether or not output has been produced; reading $\perp$ means that no output was produced, we do not consider delayed outputs (where $\perp$ could mean 'no output *yet*').

3.4 Algorithm for IOPOA Conformance Testing

Single State Identifying Sequence: The implementation can be said to have implemented a state $\mathbf{s}_k$ if it can be shown that there is a state in the implementation that behaves like $\mathbf{s}_k$ when the input ξ_k is entered. The state $\mathbf{s}_k$ has been *identified* in the implementation. As already pointed out, the difficulty lies in the inter-channels causal relationships: We can easily observe that $\lambda(\mathbf{s}_k, \xi_k)$ is produced by the implementation, but checking that $\omega(\mathbf{s}_k, \xi_k)$ is correct requires more work.

Theorem 2. *An implementation of an IOPOA, assumed to be in a state $\mathbf{s}_k$ for which ξ_k is a distinguishing sequence, can be verified to have implemented $\mathbf{s}_k$ with the following test sequence:*

$$\left[\Delta_1\left(\xi_k\right) \circ \tau_{\mathbf{s}_k}^{\xi_k}\right]^n \circ \left[\Delta_2\left(\xi_k\right) \circ \tau_{\mathbf{s}_k}^{\xi_k}\right]^n \circ \ldots \circ \left[\Delta_p\left(\xi_k\right) \circ \tau_{\mathbf{s}_k}^{\xi_k}\right]^n, \tag{8}$$

where $[I]^n$ stands for the application of input sequence I n times.

Proof. By assumption, the IOPOA is deterministic and the implementation has at most n states. Thus, after entering the same input n times, the implementation is necessarily "locked" in a cycle of states and will not leave that cycle while the same input is entered. The input sequence will thus clearly loop between states that output $\lambda(\mathbf{s}_k, \xi_k)$ when input ξ_k. There are between 1 and n such states. By entering $[\Delta_i(\xi_k).\tau(\delta(\mathbf{s}_k, \xi_k), \mathbf{s}_k)]^n$ for some $i \in [1, \ldots, p]$, we can verify that the (at most n) states we are looping through do exhibit the correct causal relationships on port i. Since we test all ports, at the end of the test sequence we have identified in the implementation between 1 and n states that produce $\lambda(\mathbf{s}_k, \xi_k)$ and $\omega(\mathbf{s}_k, \xi_k)$ when input ξ_k.

Denote by $\Gamma(\mathbf{s}_i)$ the input sequence (8) for state $\mathbf{s}_i$. When adaptive distinguishing sequences exist, it is possible to find one of size $O(n^2)$ [21]. Moreover, transfer sequences have size $O(n)$, so the entire test sequence is of size $O(pn^3)$ when using adaptive distinguishing sequence.

3.5 Checking Sequence Construction

As a direct consequence of Theorem 2, it is easy to see that, assuming that the machine starts in its initial state $\mathbf{s^{in}} = \mathbf{s}_1$ and that $\{\xi_1, \ldots, \xi_n\}$ is a set of adaptive distinguishing sequences, the following test sequence checks that the implementation has n states, each of which reacts correctly when input the corresponding distinguishing sequence:

$$\Gamma(\mathbf{s}_1) \circ \tau(\mathbf{s}_1, \mathbf{s}_2) \circ \Gamma(\mathbf{s}_2) \circ \tau(\mathbf{s}_2, \mathbf{s}_3) \circ \ldots \circ \Gamma(\mathbf{s}_n) \circ \tau(\mathbf{s}_n, \mathbf{s}_1) \circ \Gamma(\mathbf{s}_1) \qquad (9)$$

When using adaptive checking sequences, this test sequence is of size $O(pn^4)$, since we have seen that a state identification sequence $\Gamma(\mathbf{s}_i)$ can be executed in size $O(pn^3)$ and a transfer sequence in $O(n)$, and since we have n states to verify.

In order to check the transitions, let us assume that in the IOPOA we have $\mathbf{x} \in \mathcal{X}, \mathbf{s}_i, \mathbf{s}_j \in S$ such that $\mathbf{s}_j = \delta(\mathbf{s}_i, \mathbf{x})$. We need to test that when the implementation is in a state identified as $\mathbf{s}_i$, if $\mathbf{x}$ is input the implementation outputs $\lambda(\mathbf{s}_i, \mathbf{x})$ to move into a state identified as $\mathbf{s}_j$, while respecting $\omega(\mathbf{s}_i, \mathbf{x})$.

The test sequence $\Gamma(\mathbf{s}_i).x.\Gamma(\mathbf{s}_j)$ can be used to check the transition's end states and $\lambda(\mathbf{s}_i, \mathbf{x})$. In order to verify $\omega(\mathbf{s}_i, \mathbf{x})$, the test sequence (5) can be used, but we have to ensure that $\tau(\delta(\mathbf{s}_i, \mathbf{x}), \mathbf{s}_i)$ brings indeed the implementation back to a state identified as $\mathbf{s}_i$, which can be achieved by the test sequence $\Gamma(\mathbf{s}_i).x.\tau(\delta(\mathbf{s}_i, \mathbf{x}), \mathbf{s}_i).\Gamma(\mathbf{s}_i)$. So, writing $\tau^{\mathbf{x}}_{\mathbf{s}_i} = \tau(\delta(\mathbf{s}_i, \mathbf{x}), \mathbf{s}_i)$, the entire test of the transition $\mathbf{x}$ can be done with the test sequence:

$$\Gamma(\mathbf{s}_i) \circ x \circ \Gamma(\mathbf{s}_j) \circ \tau^{\mathbf{x}}_{\mathbf{s}_i} \circ \Gamma(\mathbf{s}_i) \circ \Delta_1(\mathbf{x}) \circ \tau^{\mathbf{x}}_{\mathbf{s}_i} \circ \Delta_2(\mathbf{x}) \circ \tau^{\mathbf{x}}_{\mathbf{s}_i} \circ \ldots \circ \Delta_p(\mathbf{x}) \circ \tau^{\mathbf{x}}_{\mathbf{s}_i} \quad (10)$$

When using adaptive checking sequences, this test sequence is of size $O(pn^3)$. It must be done for every transition. If we assume t transitions, and since it is possible to go from any state to any other state in $O(n)$, testing every transition can be done in $O(tpn^3)$. The following result immediately follows:

Theorem 3. *Given an IOPOA of n states and t transitions having an adaptive checking sequence, assuming that the implementation is in the initial state, the following test sequence is a checking sequence of size $O(tpn^3 + pn^4)$:*

1. *Check all states with the test sequence (9)*
2. *For all transitions do:*
   (a) *transfer to the starting state of the transition*
   (b) *check the transition with the test sequence (10)*

Note that given that the IOPOA modeling leads to an exponential reduction of the size of the model compared to the multiport deterministic model, the checking sequence constructed with theorem 3 is also considerably shorter than one for a multiport deterministic model of the same system when dealing with sufficiently large and concurrent systegms.

4 Extensions and Outlook

4.1 Conformance Testing for Automata Without Distinguishing Sequences

Not every automaton has distinguishing sequences. In the absence of distinguishing sequences, one can always create checking sequences based on *separating families of sequences*. Adapting the definition of [17] to IOPOAs, a separating family of sequences for an IOPOA is a collection of n sets $Z_1, Z_2, \ldots, Z_n$, one collection per state. Each collection is made of up to $n-1$ sequences, such that for every pair of states $\mathbf{s}_i, \mathbf{s}_j \in S$, there is an input string α such that α is a prefix of some sequence of Z_i and of some sequence of Z_j and $\lambda(\mathbf{s}_i, \alpha) \neq \lambda(\mathbf{s}_j, \alpha)$ or $\omega(\mathbf{s}_i, \alpha) \neq \omega(\mathbf{s}_j, \alpha)$.

Separating families always exist for minimized automata, and can be used to create checking sequences based on *identifying sequences*; these checking sequences are in the worst case of exponential length in the number of separating sequences.

The construction carries over to the IOPOA case; the details will be given in an extended version of the present paper.

4.2 State Identification and State Verification

The *state identification* and *state verification* problems are two common and well known questions: find out the state the implementation is in (state identification) and verify that the implementation is indeed in a given state (state verification). With sequential input automata, the former can be answered if the automata has a distinguishing sequence, while the latter can be answered if the state has a *unique input output (UIO)*.

Unfortunately, neither questions can be answered with IOPOAs, even with a distinguishing sequence. The problem lies again in the inter-channel causal relationships that cannot be directly observed, and yet can be the only difference

between two states. In order to uncover these differences, several tests of the states can be necessary, which is simply impossible when the state is unknown or unsure.

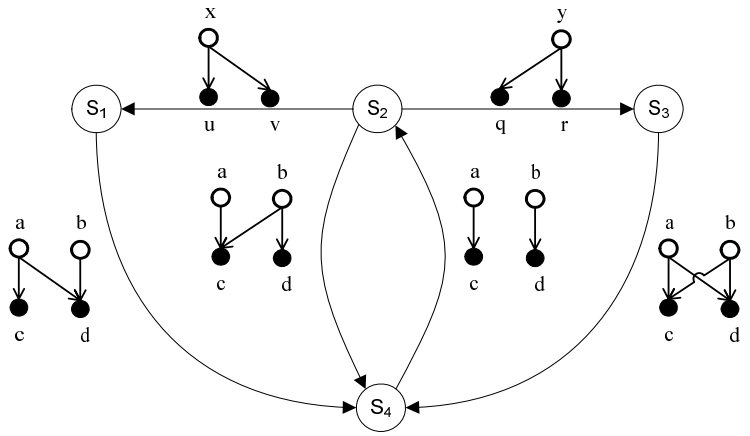

Fig. 3. An IOPOA for which states can neither be identified nor verified

The figure 3 illustrate the problem. In this IOPOA, the simple input (a, b) is a distinguishing sequence. Yet, the only strategies, a then b or b then a cannot distinguish between states s_2 and s_3 or s_1 and s_3 respectively. And since, whatever strategy, the implementation should be in state s_4 afterward, it is not possible to extend the test any further to clarify the situation, and thus state identification is not possible. For the same reason, it is not possible to ensure that the implementation is currently in state s_3.

4.3 Homing and Synchronizing Sequences

As opposed to the state identification and verification problems outlined in Section 4.2, homing sequences and synchronizing sequences are not difficult with IOPOAs.

A synchronizing sequence is a sequence that always takes the implementation to a particular state regardless of the state it was in when the sequence was entered. Clearly, not every automaton has such a synchronizing sequence. On the other hand, a synchronizing sequence does not involve any observation of the outputs of the implementation. Thus, if such a sequence exists, it can be used even with an IOPOA.

A *homing* sequence has the weaker property of taking the implementation to some known state, although not necessarily the same state depending on the unknown initial state. If the automaton is reduced, then such a homing sequence necessarily exists. In this case, the output plays a key role, since allows us to know the ending state. Yet, the classical way of constructing such an homing

sequence is to pick two random states and build a sequence that tells them apart (such a sequence always exists in a reduced machine), and keep going until we have told all states pairwise apart. This can be easily achieved with IOPOAs, even if the two states differ only by non directly observable inter channels causal relationships, since we know what we are trying to uncover, and we can thus test for it. As an example, consider the IOPOA of Figure 4. Initially, we do not know the current steate, so it could be $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4\}$. Say we want to separate $\mathbf{s}_1$ from $\mathbf{s}_2$; this can been done by delaying input a and observe whether d is output. Thus, the input sequence $< \perp, b >, < a, \perp >$ will generate either $< \perp, \perp >< c, d >$ or $< \perp, d >< c, \perp >$. In the first case, we were on $\mathbf{s}_1$ or $\mathbf{s}_4$, and we are now on $\{\mathbf{s}_2, \mathbf{s}_1\}$, and in the other case we were on $\mathbf{s}_2$ or $\mathbf{s}_3$, and we are now on $\{\mathbf{s}_3, \mathbf{s}_4\}$. The very same input again will tell apart the elements of these two sets.

So, the homing sequence is $< \perp, b >, < a, \perp >, < \perp, b >, < a, \perp >$, and the interpretation of the observation is, for the final state:

$$< \perp, \perp >< c, d >< \perp, \perp >< c, d > \Rightarrow \mathbf{s}_2$$
$$< \perp, \perp >< c, d >< \perp, d >< c, \perp > \Rightarrow \mathbf{s}_3$$
$$< \perp, d >< c, \perp >< \perp, \perp >< c, d > \Rightarrow \mathbf{s}_1$$
$$< \perp, d >< c, \perp >< \perp, d >< c, \perp > \Rightarrow \mathbf{s}_4$$

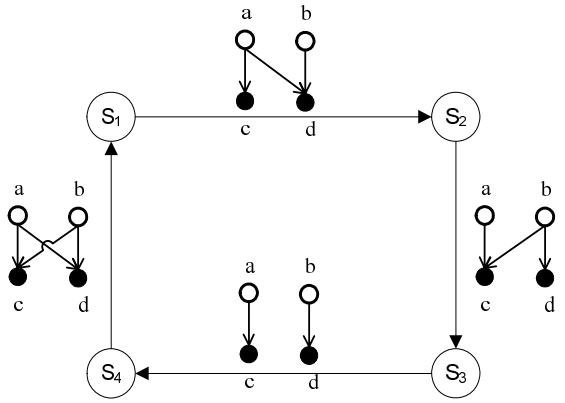

Fig. 4. Homing sequences can be found for IOPOA

5 Conclusion

We have introduced a generalized testing framework that includes and generalizes the classical I/O automaton setup. Using I/O partial order automata, asynchrony in inputs can be easily and concisely specified. Where a listing of all possible combinations of concurrent inputs is required with the Multiports Deterministic FSM model usually seen in the literature, a single transition is necessary with I/O partial order automata, leading to a model that can be exponentially smaller. I/O partial order automata allow also to specify the causal

order between inputs and outputs, including unobservable interprocess causal relationships.

We have provided a test method to check the correctness of an implementation for a specification provided with an I/O partial order automata that has an adaptive distinguishing sequence. We show that in this case, we can produce a checking sequence of polynomial size in the number of transitions and the number of ports, thus we are not "paying back" the exponential reduction achieved by the model. This non intuitive result shows that I/O partial order automata are a powerful model when it comes to specifying and testing concurrency in distributed systems.

References

1. Chen, J., Hierons, R., Ural, H.: Conditions for resolving observability problems in distributed testing. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 229–242. Springer, Heidelberg (2004)
2. Chen, X.J., Hierons, R.M., Ural, H.: Resolving observability problems in distributed test architecture. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 219–232. Springer, Heidelberg (2005)
3. Sarikaya, B., Bochmann, G.v.: Synchronization and specification issues in protocol testing. IEEE Transactions on Communications 32, 389–395 (1984)
4. Luo, G., Dssouli, R., Bochmann, G.V., Venkataram, P., Ghedamsi, A.: Test generation with respect to distributed interfaces. Comput. Stand. Interfaces 16, 119–132 (1994)
5. Tai, K., Young, Y.: Synchronizable test sequences of finite state machines. Computer Networks and ISDN Systems 30, 1111–1134 (1998)
6. Hierons, R.M.: Testing a distributed system: Generating minimal synchronised test sequences that detect output-shifting faults. Information and Software Technology 43, 551–560 (2001)
7. Khoumsi, A.: A temporal approach for testing distributed systems. Software Engineering, IEEE Transactions on 28, 1085–1103 (2002)
8. Wu, W.J., Chen, W.H., Tang, C.Y.: Synchronizable for multi-party protocol conformance testing. Computer Communications 21, 1177–1183 (1998)
9. Cacciari, L., Rafiq, O.: Controllability and observability in distributed testing. Inform. Software Technol. 41, 767–780 (1999)
10. Boyd, S., Ural, H.: The synchronization problem in protocol testing and its complexity. Information Processing Letters 40, 131–136 (1991)
11. Dssouli, R., von Bochmann, G.: Error detection with multiple observers. In: Protocol Specification, Testing and Verification, vol. V, pp. 483–494. Elsevier, North Holland (1985)
12. Dssouli, R., von Bochmann, G.: Conformance testing with multiple observers. In: Protocol Specification, Testing and Verification, vol. VI, pp. 217–229. Elsevier, North Holland (1986)
13. Rafiq, O., Cacciari, L.: Coordination algorithm for distributed testing. The. Journal of Supercomputing 24, 203–211 (2003)
14. Hierons, R.M., Ural, H.: Uio sequence based checking sequence for distributed test architectures. Information and Software Technology 45, 798–803 (2003)
15. Chen, J., abd, H.U., R.M.H.: Overcoming observability problems in distributed test architectures (Information Processing Letters) (to appear)

16. Jourdan, G.V., Ural, H., Yenigün, H.: Minimizing coordination channels in distributed testing. In: Najm, E., Pradat-Peyre, J.F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 451–466. Springer, Heidelberg (2006)
17. Lee, D., Yannakakis, M.: Principles and methods of testing finite–state machines – a survey. In: Proceedings of the IEEE, vol. 84, pp. 1089–1123 (1996)
18. Gill, A.: Introduction to The Theory of Finite State Machines. McGraw Hill, New York (1962)
19. Hennie, F.C.: Fault–detecting experiments for sequential circuits. In: Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design, Princeton, New Jersey, pp. 95–110 (1964)
20. Lee, D., Yannakakis, M.: Testing finite state machines: state identification and verification. IEEE Trans. Computers 43, 306–320 (1994)
21. Sokolovskii, M.N.: Diagnostic experiments with automata. Journal Cybernetics and Systems Analysis 7, 988–994 (1971)