Stefan Haar\textsuperscript{1}  Claude Jard\textsuperscript{2}  Guy-Vincent Jourdan\textsuperscript{3}

\textsuperscript{1}IRISA, INRIA  
Rennes, France  
Stefan.Haar@irisa.fr

\textsuperscript{2}IRISA, ENS Cachan Bretagne  
Rennes, France  
Claude.Jard@bretagne.ens-cachan.fr

\textsuperscript{3}University of Ottawa, SITE  
Ottawa, Canada  
gvj@site.uottawa.ca

TestCom-FATES 2007
Outline

1. Introduction
2. The model
3. Testing IOPOAs
4. Other testing questions
5. Further Research
General problem

The specification

- We consider that a formal specification of the system is provided
- The specification is given via an automata
- We are provided with an implementation of the specification, which may or may not be correct
The test

- The implementation is a black box with which we can interact
- We want to automatically generate test sequences to prove properties such as:
  - The conformance of the implementation to the specification
  - Homing and synchronizing sequences
  - State identification and State Verification
Sequential Systems

Definition

A deterministic FSM $M$ is defined by a tuple $(S, s_1, X, Y, \delta, \lambda)$

- $S$ is a finite set of states
- $s_1 \in S$ is the initial state
- $X$ is the finite input alphabet
- $Y$ is the finite output alphabet
- $\delta : S \times X \rightarrow S$ is the next state function
- $\lambda : S \times X \rightarrow Y$ is the output function

In the following, we assume that our FSMs are deterministic, complete and minimized.
Sequential Systems

Figure: The FSM $M_0$

In $M_0$

- $S = \{s_1, s_2, s_3, s_4, s_5\}$
- $X = \{a, b\}$
- $Y = \{0, 1\}$
**Definition**

A \(p\)-multiport deterministic FSM \(M\) is defined by a tuple \((S, s_1, p, X_1, X_2, \ldots, X_p, \delta, Y_1, Y_2, \ldots, Y_p, \lambda_1, \lambda_2, \ldots, \lambda_p)\)

- \(S\) is a finite set of *n states*
- \(s_1 \in S\) is the *initial state*
- \(X_i\) is the set of input symbols on port \(i\) such that for \(j \in [1 \ldots p]\) and \(j \neq i\), \(X_i \cap X_j = \emptyset\).
- \(Y_i\) is the set of output symbols on port \(i\) such that for \(i, j \in [p]\) if \(i \neq j\) then \(Y_i \cap Y_j = \{-\}\), where \(-\) is null output.
- \(\delta : S \times X \rightarrow S\) is the *next state function.*
- \(\lambda_i : S \times X \rightarrow Y_i\) is the *output function on port \(i\).*
Concurrent multi-port machines

Figure: The multiports FSM $M_0$
Concurrent multi-port machines

In $M_0$

- $S = \{s_1, s_2, s_3, s_4, s_5\}$
- $X_1 = \{a\}$, $X_2 = \{b\}$, $X_3 = \{c\}$
- $Y_1 = \{1\}$, $Y_2 = \{2\}$, $Y_3 = \{3\}$
Limitations with multi-port machines

One problem with p-multiport I/O Automata is that despite the fact that they are meant to specify distributed systems, they do it in a sequential way, one input at the time. This leads to a model that is

- **Inefficient**: every possible combination of concurrent inputs must be specified in the model
- **Unclear**: the causal relationships between inputs and outputs is not explicitly described
Concurrent multi-port machines

Figure: (Partial) Multiports Deterministic FSM.
A new model

We need 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.
IO-PO-automata

Figure: The IOPOA corresponding to the multiports deterministic FSM $M_0$. 
IO-PO-automata

Definition

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

- $S$ is a finite set of *states* and $s_1 = s^\text{in} \in S$ is the *initial state*;
- $\text{Chn} = \pi_1, \ldots, \pi_p$ is the set of *I/O channels* (ports),
- $\mathcal{I}$ is the common *input alphabet*, and $\mathcal{O}$ the common *output alphabet* for all channels.
- $\delta : S \times \mathcal{X} \to S$ is a (partial) next state function
- $\lambda : S \times \mathcal{X} \to \mathcal{Y}$ is the output function
- $\ldots$
IO-PO-automata

**Definition**

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

- ...  
- $\omega$ is a PO transition label function: For any $(s, x) \in S \times \mathcal{X}$ such that $\delta(s, x) = s'$ and $\lambda(s, x) = y \in \mathcal{Y}$, $\omega(s, x) \subseteq (\{x_1, \ldots, x_p\} \times \{y_1, \ldots, y_p\})$ is a partial order that satisfies
  - $x_i < y_i$ for all $i \in \{1, \ldots, p\}$ such that $x_i \neq \perp$ and $y_i \neq \perp$, and
  - if $x_i = \perp$, then $x_i \not\leq y_j$ for all $j \in \text{Chn}$. 
IO-PO-automata

Strengths of the model

IOPOA do provide, as a model, a great improvement over p-multiport IO Automata in terms of

- **Size**: a single transition (a single order) can express a large number of transitions in the multiport model
- **Clarity**: causal relationships between input and outputs are explicitly described and do not have to be “guessed” by the implementer

But... What about testing? Are we “paying back” the efficiency of the model when generating test cases, with test sequences that are (exponentially) longer than the ones produced with multiport I/O?
Definition (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 $i, j \in [1, \ldots, n]$, $i \neq j$, $\xi_i$ and $\xi_j$ have a non-empty common prefix $\xi_{ij}$ and $\lambda(s_i, \xi_{ij}) \neq \lambda(s_j, \xi_{ij})$ or $\omega(s_i, \xi_{ij}) \neq \omega(s_j, \xi_{ij})$. 
Definition (Checking Sequence)

Let $M_1 = (S_1, s^\text{in}_1, \text{Chn}, \mathcal{I}, \mathcal{O}, \delta_1, \lambda_1, \omega_1)$ be an IOPOA. A checking sequence of $M_1$ is an input sequence $I$ which distinguishes $M_1$ from any IOPOA $M_2 = (S_2, s^\text{in}_2, \mathcal{I}, \mathcal{O}, \text{Chn}, \delta_2, \lambda_2, \omega_2)$ in $C(M_1)$ that does not conform to $M_1$, i.e. such that $\forall s \in S_2$, $\lambda_1(s^\text{in}_1, I) \neq \lambda_2(s, I)$ or $\omega_1(s^\text{in}_1, I) \neq \omega_2(s, I)$. 
Sequential Input Automata Testing

**States verification**

Assuming that the machine starts in its initial state $s_{in} = s_1$, the following test sequence checks that the implementation has $n$ states, each of which reacts correctly when input the distinguishing sequence for that state:

$$\xi_1 \circ \tau(\delta(s_1, \xi_1), s_2) \circ \xi_2 \circ \tau(\delta(s_2, \xi_2), s_3) \circ \ldots \circ \xi_n \circ \tau(\delta(s_n, \xi_n), s_1) \circ \xi_1$$

**Transitions verification**

Testing a transition $a/b$ going from $s_i$ to $s_j$, assuming the implementation is in a state $s_k$:

$$\tau(s_k, s_{i-1}) \circ \xi_{i-1} \circ \tau(\delta(s_{i-1}, \xi_{i-1}), s_i) \circ a \circ \xi_j$$
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

**Solution**
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

**Solution**
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

**Solution**
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
Testing IOPOA

The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

**Solution**
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

**Solution**
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

Solution
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
The “classical” approach to testing automata does not work with IPPOA because the causal relationships are not observed.

**Solution**
- Delay input on one port
- Observe outputs
- Send last input
- Observe outputs
- repeat
Delayed input testing

We define input vector $\mathbf{x}^i$ as

$$
\mathbf{x}_j^i \triangleq \begin{cases} 
\bot & : i = j \\
\mathbf{x}_j & : i \neq j,
\end{cases}
$$

$$
\mathbf{x}_j^i \triangleq \begin{cases} 
\mathbf{x}_i & : i = j \\
\bot & : i \neq j.
\end{cases}
$$

On a sequence $\alpha = \alpha_1 \ldots \alpha_n$

$$
\Delta_i(\alpha) \triangleq \hat{\alpha}_1^i \hat{\alpha}_1^i \hat{\alpha}_2^i \hat{\alpha}_2^i \ldots \hat{\alpha}_n^i \hat{\alpha}_n^i
$$
State verification

Theorem

An implementation of an IOPOA, assumed to be in a state $s_k$ for which $\xi_k$ is a distinguishing sequence, can be verified to have implemented $s_k$ with the following test sequence:

$$\Gamma(s_k) = \left[ \Delta_1(\xi_k) \circ \tau_{s_k}^{\xi_k} \right]^n \circ \left[ \Delta_2(\xi_k) \circ \tau_{s_k}^{\xi_k} \right]^n \circ \ldots \circ \left[ \Delta_p(\xi_k) \circ \tau_{s_k}^{\xi_k} \right]^n$$

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

Checking all states

$$\Gamma(s_1) \circ \tau(s_1, s_2) \circ \Gamma(s_2) \circ \tau(s_2, s_3) \circ \ldots \circ \Gamma(s_n) \circ \tau(s_n, s_1) \circ \Gamma(s_1)$$

Checking transitions

$$\Gamma(s_i) \circ x \circ \Gamma(s_j) \circ \tau^x_{s_i} \circ \Gamma(s_i) \circ \Delta_1(x) \circ \tau^x_{s_i} \circ \Delta_2(x) \circ \tau^x_{s_i} \circ \ldots \circ \Delta_p(x) \circ \tau^x_{s_i}$$
Theorem

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 $O(tpn^3 + pn^4)$
Figure: An IOPOA for which states can neither be identified nor verified.
Homing and Synchronizing sequences

Figure: Homing sequences can be found for IOPOA.
Further research directions

- Checking sequences for IOPOA without distinguishing sequences
- IOPOA with arbitrary partial order
- Petri Nets
- Sufficient conditions for weak synchronization at states
- On-going implementation