Controllability Through Nondeterminism in Distributed Testing

Robert Hierons, Mercedes Merayo, Manuel Núñez

Controllability through nondeterminism in distributed testing*

Robert M. Hierons[1], Mercedes G. Merayo[2], and Manuel Núñez[2]

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

Abstract. If the system under test interacts with its environment at physically distributed ports, there is a separate independent tester at each port, and there is no global clock then we are testing in the distributed test architecture. It is known that the distributed test architecture can lead to additional controllability problems in which a tester cannot know when to send an input and this has led to most test generation techniques aiming to produce controllable test cases. However, there may be no controllable test case that achieves a given objective. This paper introduces the notion of a test section, in which each tester has a fixed input sequence to apply and there is no attempt to synchronise the testers. It defines the notion of a test section being convergent and shows how convergent test sections can can be used as the basis of a less restrictive form of controllability.

1 Introduction

Software testing has traditionally been represented as a process in which a single tester synchronously interacts with the system under test (SUT). However, testing does not operate in this way if the SUT has multiple physically distributed interfaces (ports) at which it interacts with its environment; one might then have one local tester at each interface. For example, when testing the implementation of a layer of a communications protocol there might be one local tester that acts as the layer above the SUT and a second local tester that sits on a different machine [21, 4, 5]. More generally, if the SUT has multiple ports then there might be a separate tester at each port. If these testers do not synchronise their actions and there is no global clock then we are testing in the ISO standardised distributed test architecture [14].

Most work on formal testing in the distributed test architecture uses *multi-port finite state machine (FSM)* models [21, 4, 5] in which a transition is triggered

* Research partially supported by the projects DArDOS (TIN2015-65845-C3-1-R (MINECO/FEDER)) and SICOMORo-CM (S2013/ICE-3006).

by an input, produces up to one output at each port, and possibly changes the state. We also use this approach, of assuming that the specification is a multi-port FSM, and we use the term FSM for such models. Note, however, that some work has explored more general types of models in which, for example, a transition can be labelled by a partially-ordered multi-set of actions [7, 1, 18, 19].

Previous work has shown that the distributed test architecture changes the nature of testing. Let us suppose that we wish to start a test sequence with input x_1 at port 1, this should lead to output y_1 at port 1 and we wish to follow this with input x_2 at port 2. We might implement this using a test case t in which the tester t_1 at port 1 applies x_1 and the tester t_2 at port 2 applies x_2. Since we are testing in the distributed test architecture, tester t_2 does not observe the input or output at port 1 and so cannot know when to supply x_2. Thus, if we use the test case t then we cannot guarantee that the inputs arrive in the correct order; this introduces non-determinism into testing even if the SUT is deterministic. This situation is normally called a *controllability problem* [21, 4, 5]; if a test case has no controllability problems then it is *controllable*. Controllability problems can lead to situations in which we cannot know whether a test objective has been achieved and also make it more difficult to debug a faulty system and trace failures back to requirements. As a result, almost all work in distributed testing aims to produce controllable test cases (see, for example, [21, 4, 5, 16, 23, 13]).

While there are test generation algorithms that produce controllable test cases from FSMs, these have inherent limitations. In particular, one can construct an FSM M such that controllable testing can achieve very little. Consider, for example, the fragment of an FSM shown in Figure 1. Here, the label $x_p/(y_q, y_r)$ on an arc means that the input is x_p at port p and the output is y_q at port q and y_r at port r, with $-$ denoting no output at the corresponding port. If an input sequence starts with x_2 then there is no change in state and the resultant output is at port 2 only. Thus, for a test sequence to be controllable we require that the next input is at port 2, since only the tester at port 2 observed the previous input and output. It is straightforward to see that this situation continues and so any controllable test case that starts with x_2 cannot contain x_1 and only visits state s_0. If we now consider a test case that starts with x_1, the first input takes the FSM to state s_1 and produces y_1 at port 1 only. Therefore, for a test sequence to be controllable, the next input must be at port 1. However, if we apply x_1 then the FSM returns to s_0 and produces output at port 1 only. Thus, any controllable test case that starts with x_1 cannot contain x_2 and only visits s_0 and s_1. Hence, if an FSM is of the form shown in Figure 1 then controllable testing can only visit s_0 and s_1 irrespective of how many states the FSM has.

There are several ways in which one might try to tackle the above problem. One approach is for the testers to synchronise actions through message exchange [2, 20]. When feasible, this allows controllability problems to be overcome and provides a general solution. However, this requires a network to be introduced and so can make testing more expensive. Message latency might also lead to situations in which a test case cannot be executed since it has timing

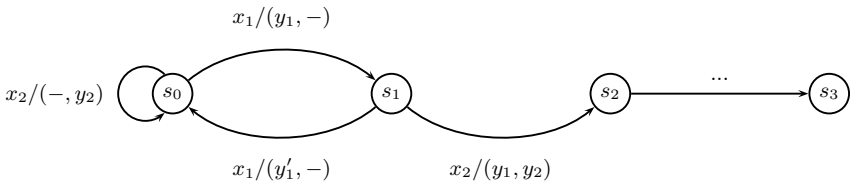


Fig. 1: Part of an FSM where controllable testing achieves little

constraints. A second line of work aims to allow one to reason about what can be achieved in controllable testing [8]. In particular, it is possible to construct an FSM $\chi_{min}(M)$ from the specification FSM M such that the transitions of $\chi_{min}(M)$ are those that can be executed in controllable testing and it is possible to construct a non-deterministic FSM $\chi_{max}(M)$ such that controllable testing can show that an SUT is faulty if and only if there are traces of the SUT that are not in the language defined by $\chi_{max}(M)$. One can use $\chi_{min}(M)$ and $\chi_{max}(M)$ to reason about the potential effectiveness of controllable testing. If the tester decides that controllable testing is sufficiently powerful then they can use a recently developed technique that generates a test suite that achieves as much as possible given the constraint that testing is controllable [10]. It is also possible to abandon the restriction that we use controllable test cases. However, as noted above, there are good practical reasons for using controllable test cases and it has also been shown that test generation problems, such as finding a prefix of a test case that is guaranteed to take M to a given state s, become undecidable [9].

Consider now the part of an FSM, with three ports, shown in Figure 2. If testing starts with input x_1 then a controllable test case can then apply input at any port. There are two paths that take the FSM from s_1 to s_4: one has label $x_2/(y_1, y_2, -)x_3/(y_1, y'_2, y'_3)$ and the other has label $x_3/(-, -, y_3)x_2/(y_1, y'_2, y'_3)$. Both of these are uncontrollable: in the first case the tester at port 3 does not observe input or output from the transition with label $x_2/(y_1, y_2, -)$ and in the second case the tester at port 2 does not observe input or output from the transition with label $x_3/(-, -, y_3)$. However, if we just require that the tester at port 2 sends input x_2 and the tester at port 3 sends input x_3 then state s_4 is reached irrespective of the order in which the inputs are supplied. Thus, even though a corresponding test case is not controllable, we do know that it reaches s_4, with this situation being similar to partial order reduction (see, for example, [6]). In addition, the testers at ports 2 and 3 know when s_4 has been reached since at this point they receive particular outputs (y'_2 at port 2, y'_3 at port 3). Testing can thus continue with one of these testers applying an input in state s_4. In contrast, if one considers the two paths then in one case the tester at port 1 observes y_1 and in the other the tester at port 1 observes y_1y_1. If the tester at port 1 observes y_1 then there are two possible explanations and the state is either s_2 or s_4. As a result, one cannot guarantee that the tester at port 1 knows when s_4 has been reached. This paper formalises and extends these ideas, showing how one can relax controllable testing while retaining some of its benefits.

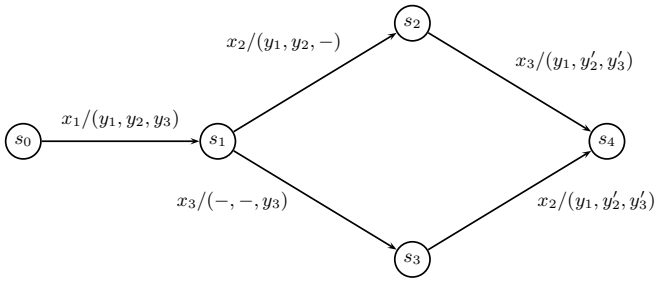


Fig. 2: Part of an FSM with controllability problems

The rest of the paper is structured as follows. We start in Section 2 by defining FSMs and the notation used. Section 3 shows how we can relax the notion of controllability. Section 4 then considers computational complexity issues and a bounded form. Finally, Section 5 draws conclusions and discusses related work.

2 Preliminaries

This paper concerns the testing of a state-based system and, as such, we will reason about sequences of *inputs* and *outputs*. In testing the SUT will receive a sequence of inputs and there will be a resultant sequence of input/output pairs, called an *input/output sequence* or *trace*.

Definition 1. *We let X be the set of inputs of the SUT and Y the set of outputs of the SUT. Given $x \in X$ and $y \in Y$, the corresponding input/output pair x/y represents the SUT producing output y in response to input x.*

A trace is a (possibly empty) sequence of input/output pairs. The trace that has input/output pair x^1/y^1 followed by x^2/y^2, ..., and finally x^k/y^k will be represented using either $x^1/y^1\ x^2/y^2\ \ldots x^k/y^k$, $x^1x^2\ldots x^k/y^1y^2\ldots y^k$, or $\bar{x}/\bar{y}$ where $\bar{x} = x^1x^2\ldots x^k$ and $\bar{y} = y^1y^2\ldots y^k$.

Given a sequence $\bar{a}$ and an element a we let $a{\cdot}\bar{a}$ denote the sequence in which a is followed by $\bar{a}$. Given a sequence $\bar{a} = a^1\ldots a^k$, with $k \geq 0$, we will let $pre(\bar{a}) = \{a^1\ldots a^i | 0 \leq i \leq k\}$ denote the set of prefixes of $\bar{a}$ and we use ϵ to represent the empty sequence. Given a set A of sequences, $pre(A) = \bigcup_{\bar{a}\in A}\{pre(\bar{a})\}$.

Since a trace is a sequence of input/output pairs, all prefixes of traces are also traces and so

$$pre(x^1/y^1\ x^2/y^2\ \ldots x^k/y^k) = \{x^1/y^1\ x^2/y^2\ \ldots x^i/y^i | 0 \leq i \leq k\}$$

Work on testing from an FSM in the distributed test architecture has used multi-port FSMs. In such an FSM, there is a finite set of ports, which represent the interfaces at which the SUT interacts with its environment. We let P denote the set of (m) ports, with $\{1, \ldots, m\}$ denoting the names of the ports. If an

input is received in a multi-port FSM then this triggers a transition, which can lead to a change in state and at most one output being produced at each port.

Definition 2. *A multi-port FSM M with m ports is defined by a tuple $(S, s_0, X, Y, \delta, \lambda)$ in which:*

- *S is the finite* set of states *of M.*
- *$s_0 \in S$ is the* initial state *of M.*
- *$X = X_1 \cup \ldots \cup X_m$ is the finite* input alphabet *of M, where for $1 \leq p \leq m$, X_p is the input alphabet at port p and for all $1 \leq p < q \leq m$ we have that $X_p \cap X_q = \emptyset$.*
- *$Y = (Y_1 \cup \{-\}) \times \ldots \times (Y_m \cup \{-\})$ is the* output alphabet *of M, where for $1 \leq p \leq m$, Y_p is the output alphabet at port p, $-$ denotes no output, and for all $1 \leq p < q \leq m$ we have that that $Y_p \cap Y_q = \emptyset$. In addition, the inputs and outputs are disjoint and so $X \cap \cup_{1 \leq p \leq m} Y_p = \emptyset$.*
- *δ is the (total) next state function of type $S \times X \to S$.*
- *λ is the (total) output function of type $S \times X \to Y$.*

If M receives input x when in state s then it moves to state $s' = \delta(s, x)$ and outputs an m-tuple $y = \lambda(s, x)$. This defines a transition *$t = (s, s', x/y)$. We let T denote the set of transitions of M. When we refer to actions, a subscript will denote the port at which it is observed and a superscript will denote its position in a sequence.*

The functions δ and λ can be extended in the usual way to deal with sequences of inputs. Specifically, given a state $s \in S$ and a sequence of inputs $\bar{x} = x^1 x^2 \cdots x^n$ we define $\delta(s, \bar{x})$ as $\delta(\delta((\cdots \delta(\delta(s, x^1), x^2) \ldots), x^{n-1}), x^n)$, that is, the state reached after following the sequence $\bar{x}$ and we define $\lambda(s, \bar{x})$ as $\lambda(s, x^1) \cdot \lambda(\delta(s, x^1), x^2) \cdot \ldots \lambda(\delta((\cdots \delta(\delta(s, x^1), x^2), \ldots), x^{n-1}), x^n)$, that is, the sequence of tuples of outputs observed after following the sequence $\bar{x}$.

A path *of M is a sequence $\rho = (s_1, s_2, x_1/y_1)(s_2, s_3, x_2/y_2) \ldots (s_k, s_{k+1}, x_k/y_k)$ of consecutive transitions. We let $x_1/y_1 x_2/y_2 \ldots x_k/y_k$ denote the* label *of ρ.*

The requirement that the alphabets at the ports are pairwise disjoint is not a restriction since one can label inputs and outputs with port numbers. We will use the term FSM for multi-port FSMs and the term single-port FSM for FSMs with one port. Note that our FSMs are deterministic: the current state and input received uniquely determine the next state and output produced. Most work on testing from single-port FSMs has concerned such deterministic machines (see, for example, [3, 15, 17]), as has almost all work on distributed testing from FSMs (see, for example, [21, 4, 5, 16, 23, 13]).

Next we introduce notation to project the actions of an input sequence or a trace onto a port.

Definition 3. *Given a sequence $\bar{x} \in X^*$ and a port p, the projection $\pi_p(\bar{x})$ of $\bar{x}$ at port p can be inductively defined as follows:*

$$\pi_p(\bar{x}) = \begin{cases} \epsilon & \text{if } \bar{x} = \epsilon \\ \overline{x}' & \text{if } \bar{x} = x \cdot \overline{x}' \wedge x \notin X_p \\ x \cdot \pi_p(\overline{x}') & \text{if } \bar{z} = x \cdot \overline{x}' \wedge x \in X_p \end{cases}$$

Definition 4. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P such that $|P| = m$. Given an input/output sequence $\bar{z}$ and a port p, the projection $\pi_p(\bar{z})$ of $\bar{z}$ at port p can be inductively defined as follows:*

$$\pi_p(\bar{z}) = \begin{cases} \epsilon & \text{if } \bar{z} = \epsilon \\ \overline{z}' & \text{if } \bar{z} = x/(y_1, \ldots, y_m) \cdot \overline{z}' \wedge x \notin X_p \wedge y_p = - \\ x \cdot \pi_p(\overline{z}') & \text{if } \bar{z} = x/(y_1, \ldots, y_m) \cdot \overline{z}' \wedge x \in X_p \wedge y_p = - \\ y_p \cdot \pi_p(\overline{z}') & \text{if } \bar{z} = x/(y_1, \ldots, y_m) \cdot \overline{z}' \wedge x \notin X_p \wedge y_p \neq - \\ x \cdot y_p \cdot \pi_p(\overline{z}') & \text{if } \bar{z} = x/(y_1, \ldots, y_m) \cdot \overline{z}' \wedge x \in X_p \wedge y_p \neq - \end{cases}$$

We say that $\pi_p(\bar{z})$ is a local trace.

Given an input/output pair x/y, $ports(x/y) = \{p \in P | \pi_p(x/y) \neq \epsilon\}$ denotes the set of ports involved in x/y. Given transition $t = (s_i, s_j, x/y)$, $ports(t) = ports(x/y)$ and $port(x)$ denotes the port $p \in P$ such that $x \in X_p$.

Note that we have overloaded π_p and *ports*: the first one was previously used to project sequences of inputs and the second one denotes both the ports involved in an input/output pair and in a transition.

Let us suppose that the input sequence $x^1 \ldots x^k$ leads to output sequence $y^1 \ldots y^k$ when applied to M. In order for $x^1 \ldots x^k$ to be *controllable* [2, 22, 11] we require that the tester that applies x^i knows when to send x^i and that this is the case for all $1 < i \leq k$. If the tester at p sends x^i ($p = port(x^i)$) then it knows when to send x^i if it observed the previous transition and this is the case if either x^{i-1} is at port p or y^{i-1} has non-empty output at port p.

Definition 5. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P. Trace $x^1/y^1\ x^2/y^2\ \ldots x^k/y^k$ is* controllable *if $port(x^i) \in ports(x^{i-1}/y^{i-1})$ for all $1 < i \leq k$. Further, input sequence $x^1 \ldots x^k$ is* controllable *if $x^1 \ldots x^k/\lambda(s_0, x^1 \ldots x^k)$ is controllable and a path is controllable if its label is controllable.*

Previous work [12] showed how a directed graph $G(M)$ can be produced from FSM M such that the paths of $G(M)$, from the vertex representing the initial state of M, correspond to the controllable paths of M. The construction of $G(M)$ is based on the following concepts.

Definition 6. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P. For each state $s \in S$ and port $p \in P$ we denote by $Depart^p(s)$ the set of transitions of M whose starting state is s and whose input is at port p, that is, the set $\{(s, s', x/y) \in T | x \in X_p\}$. For each state s and set $\mathcal{P} \subseteq P$ of ports we denote by $Arrive^{\mathcal{P}}(s)$ the set of transitions whose ending state is s and that involve the set $\mathcal{P}$ of ports, that is, the set $\{(s', s, x/y) \in T | ports(x/y) = \mathcal{P}\}$.*

In order to ensure controllability, transitions belonging to $Arrive^{\mathcal{P}}(s)$ can only be followed by input at a port p if $p \in \mathcal{P}$. Thus, given transitions $\tau = (s_1, s_2, x/y)$ and $\tau' = (s_2, s_3, x'/y')$, we can follow τ by τ' without causing controllability problems if $port(x') \in ports(x/y)$. It is straightforward to see that if $\tau \in Arrive^{\mathcal{P}}(s_2)$ then we can follow τ by τ' in controllable testing if and only if

there is some $p \in \mathcal{P}$ such that $\tau' \in Depart^p(s_2)$. We will use these properties to construct the desired graph.

Definition 7. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P. The graph $G(M) = (V, E)$ provides all the controllable sequences contained in M. The vertex set of $G(M)$ is defined in two steps. First, we define an auxiliary vertex set as follows:*

1. *$v_{s_0}^P$ is in V_{aux}.*
2. *For all $s \in S$ and $\mathcal{P} \subseteq P$ we include $v_s^{\mathcal{P}}$ in V_{aux} if $Arrive^{\mathcal{P}}(s) \neq \emptyset$.*

Edge set E is defined by: for each $t = (s, s', x/y) \in T$ and $v_s^{\mathcal{P}} \in V_{aux}$ with $port(x) \in \mathcal{P}$ we include in E the edge $(v_s^{\mathcal{P}}, v_{s'}^{\mathcal{P}_t}, x/y)$ where $\mathcal{P}_t = ports(x/y)$. Finally, V is the subset of V_{aux} that includes all the nodes reachable from $v_{s_0}^P$.

The notion of path (see Definition 2) can also be used with graphs: a path is a sequence of consecutive edges. The vertex $v_{s_0}^P$ (the *initial vertex*) represents the situation in which the first input has not yet been applied; this first input can be at any port. $v_s^{\mathcal{P}}$ denotes the situation in which M has reached state s and $\mathcal{P}$ is the set of ports that can receive the next input if testing is controllable.

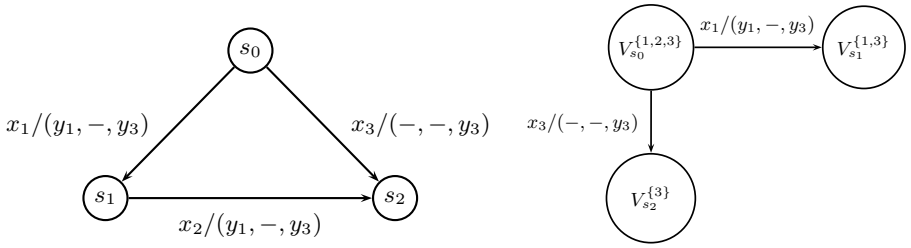
Example 1. Consider the fragment of an FSM M with port set $P = \{1, 2, 3\}$ depicted in Figure 3 (a). Figure 3 (b) shows $G(M)$. For each of the transitions that reaches a state of M, a new vertex is included in $G(M)$. For example, the state s_2 is reached by transitions $(s_1, s_2, x_2/(y_1, -, y_3))$ and $(s_0, s_2, x_3/(-, -, y_3))$. Thus, two vertexes, $V_{s_2}^{\{1,2,3\}}$ and $V_{s_2}^{\{3\}}$, respectively, are generated. The superscripts of each vertex contains the ports that are involved in the corresponding transition. The graph only contains those transitions whose input corresponds to a port included in the set associated with one of the vertices related to the outgoing state. For example, the transition $(s_1, s_2, x_2/(y_1, -, y_3))$ cannot be included in the graph because the port 2, in which the action x_2 must be applied, does not belong to the set of ports of the only vertex associated to state s_1, that is, $V_{s_2}^{\{1,3\}}$. Intuitively, a tester placed at port 2 cannot know when to apply the input x_2 because no action in the previous transition has been produced at this port. In this case we would have a controllability problem. Finally, we do not include $V_{s_2}^{\{1,2,3\}}$ because it is not reachable from $V_{s_0}^{\{1,2,3\}}$.

The following relates paths of $G(M)$ and controllable traces of M [12].

Proposition 1. *For each path ρ of M that starts at the initial state of M and has a controllable label, there is a path ρ' in $G(M)$ that starts at $v_{s_0}^P$ and has the same label. In addition, for each path ρ' of $G(M)$ that starts at $v_{s_0}^P$, there is a path ρ of M that starts at the initial state of M and has the same label.*

3 Extending the graph $G(M)$

We have seen that an FSM might have states that cannot be reached using controllable input sequences; there might be a state s such that no vertex of



(a) M (b) $G(M)$

Fig. 3: Generation of $G(M)$

the form $v_s^{\mathcal{P}}$ is reachable in $G(M)$. In this section we explain how $G(M)$ can be extended through including parts of a test that are not controllable but where the lack of controllability is not problematic. First we introduce *test sections*.

Definition 8. *Let P be a set of m ports. Given m sequences of inputs such that for all $1 \leq p \leq m$ we have that $\bar{x}_p \in X_p^*$, we say that the tuple $\overline{\overline{x}} = (\bar{x}_1, \ldots, \bar{x}_m)$ is a* test section. *Given a test section $\overline{\overline{x}} = (\bar{x}_1, \ldots, \bar{x}_m)$, we denote by $INT(\overline{\overline{x}})$ the set of interleavings of the sequences $\bar{x}_1$, ..., $\bar{x}_m$. Formally, for all $\bar{x} \in X^*$ we have that $\bar{x} \in INT(\overline{\overline{x}})$ if and only if for all $p \in P$ we have $\pi_p(\bar{x}) = \bar{x}_p$.*

We will use a double overline to denote a test section. In using a test section $\overline{\overline{x}}$, each tester simply applies its input sequence. Note that we allow empty sequences of inputs for some of the ports. We now consider conditions under which edges corresponding to test sections can be added to $G(M)$.

It is straightforward to determine which vertices of $G(M)$ can have edges labelled with a particular test section leaving them: in order to be able to apply $(\bar{x}_1, \ldots, \bar{x}_m)$ in a vertex $v_s^{\mathcal{P}}$ we require that for every $p \in P$ we have that if the tester at p is to apply input $(\bar{x}_p \neq \epsilon)$ then $p \in \mathcal{P}$.

Definition 9. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P and $\overline{\overline{x}} = (\bar{x}_1, \ldots, \bar{x}_m)$ be a test section. Let $G(M) = (V, E)$.*

Given $v_s^{\mathcal{P}} \in V$, we say that $\overline{\overline{x}}$ can be applied from *$v_s^{\mathcal{P}}$ if for all $p \in P$ we have that $\bar{x}_p \neq \epsilon$ implies that $p \in \mathcal{P}$. Given states $s, s' \in S$, we say that $\overline{\overline{x}}$ is* convergent *from s to s' if for all $\bar{x} \in INT(\overline{\overline{x}})$ we have that $s' = \delta(s, \bar{x})$. We also say that $\overline{\overline{x}}$* takes *M from s to s'. Further, we say that $\overline{\overline{x}}$ is* convergent *from s if there exists a state s' such that $\overline{\overline{x}}$ is* convergent *from s to s'.*

Convergence requires that all interleavings of the input sequences take M from state s to s'; we do not have to control which interleaving occurs. Having reached s', we might continue testing in a controllable manner.

Proposition 2. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P, $\overline{\overline{x}} = (\bar{x}_1, \ldots, \bar{x}_m)$ be a test section and $s, s' \in S$. If M is in state s, $\overline{\overline{x}}$ takes M from s to s', and from s the local tester at port p applies $\bar{x}_p$ (for all $p \in P$) then M is guaranteed to be in state s' after all inputs from $\overline{\overline{x}}$ have been received.*

If $\overline{\overline{x}}$ takes M from s to s' then there is potential to add a new edge to $G(M)$ that represents this fact. However, we then need to determine which vertex $v_{s'}^{\mathcal{P}'}$ should be reached and so the set $\mathcal{P}'$ of ports at which the next input (after the test section) can be applied. This set of ports should be the ports whose tester can determine when all of the inputs from $(\bar{x}_1, \ldots, \bar{x}_m)$ have been received. The following gives a condition under which the tester at port p can determine this.

Definition 10. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P, $\overline{\overline{x}}$ be a test section, $s \in S$ be a state of M and $p \in P$ be a port. We say that port $p \in P$ is* termination aware *when $\overline{\overline{x}}$ is applied from state s if for all $\bar{x} \in INT(\overline{\overline{x}})$ and $\bar{x}' \in pre(INT(\overline{\overline{x}})) \setminus INT(\overline{\overline{x}})$ we have $\pi_p(\lambda(s, \bar{x})) \neq \pi_p(\lambda(s, \bar{x}'))$.*

Once all inputs from $\overline{\overline{x}}$ have been received the tester at p observes a local trace of the form $\pi_p(\lambda(s, \bar{x}))$ for some $\bar{x} \in INT(\overline{\overline{x}})$; the above condition ensures that this observation cannot have been made if one or more inputs from $\overline{\overline{x}}$ have not been received. The following is clear from the previous definition.

Proposition 3. *Given FSM $M = (S, s_0, X, Y, \delta, \lambda)$ with port set P and $p \in P$, let us suppose that p is termination aware when $(\bar{x}_1, \ldots, \bar{x}_m)$ is applied from state $s \in S$. If $(\bar{x}_1, \ldots, \bar{x}_m)$ is applied from s then the tester at port p knows when all inputs from each $\bar{x}_q$ have been received.*

We can now combine the notions of convergence and termination to obtain a weaker type of controllability.

Definition 11. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P, $s, s' \in S$ be states, $\overline{\overline{x}}$ be a test section, and $\mathcal{P}, \mathcal{P}' \subseteq P$ be sets of ports. Let us suppose that $\overline{\overline{x}}$ takes M from s to s', $\mathcal{P}$ is the set of ports that are termination aware when $\overline{\overline{x}}$ is applied from state s, and $\mathcal{P}'$ is the following set of ports*

$$\mathcal{P} \cup \{p \in P | \forall \bar{x}, \bar{x}' \in INT(\overline{\overline{x}}) : \pi_p(\lambda(s, \bar{x})) = \pi_p(\lambda(s, \bar{x}'))\}$$

Then we say that $(s, \overline{\overline{x}}, s', \mathcal{P}, \mathcal{P}')$ is a semi-controllable tuple *of M. We let $Reach(M)$ be the set of semi-controllable tuples of M.*

If $(s, \overline{\overline{x}}, s', \mathcal{P}, \mathcal{P}')$ is a semi-controllable tuple then $\overline{\overline{x}} = (\bar{x}_1, \ldots, \bar{x}_m)$ is a test section with the property that if $\overline{\overline{x}}$ is applied from state s (and for all p such that $\bar{x}_p \neq \epsilon$, the tester at p knows that the state is s) then it takes M to s' and the testers in $\mathcal{P}$ are termination aware. In this definition, a port p is in $\mathcal{P}'$ if either p is termination aware or there is fixed output at p when the test section is applied. Essentially, $p \in \mathcal{P}'$ captures two scenarios that ensure that if the tester at p observes an output from a transition that is after $\overline{\overline{x}}$ then this tester can know that the output did not result from the application of the test section. We will see that this condition is important if we later wish to apply inputs at p.

$Reach(M)$ may be infinite and so an algorithm should not include a step that generates this set. Instead, in this section we assume that there is some fixed $R \subseteq Reach(M)$; this will be a parameter of the algorithms introduced. In the next section we consider the case where we place a bound k on the size

of the test sections used and so it is possible to generate the corresponding set $Reach(M, k)$.

If $(s, \overline{\overline{x}}, s', \mathcal{P}, \mathcal{P}')$ is a semi-controllable tuple, $p \in \mathcal{P}$ and $\overline{\overline{x}}$ is applied from state s then the tester at p can apply an input after $\overline{\overline{x}}$ and know that this will be received in state s'. This potentially allows an input $x_p \in X_P$ to be applied in a state s' even if $G(M)$ does not have a reachable vertex of the form $v_{s'}^{\mathcal{P}''}$ with $p \in \mathcal{P}''$. In such cases, it is possible to execute additional transitions of M in testing and to know that this has been achieved despite this not being possible in controllable testing.

We will add vertices and edges based on $R \subseteq Reach(M)$; if $v_s^{\mathcal{P}}$ is a current vertex and $(s, \overline{\overline{x}}, s', \mathcal{P}, \mathcal{P}') \in R$ then there is the potential to add a new vertex and edge if $\overline{\overline{x}}$ can be applied from $v_s^{\mathcal{P}}$ (Definition 9). Before providing an algorithm, for extending $G(M)$, we will describe two additional factors that should be considered.

Example 2. Consider again the part of an FSM shown in Figure 2. We know that (ϵ, x_2, x_3) is a test section that takes this FSM from s_1 to s_4 and also that the testers at ports 2 and 3 are termination aware. Let us suppose that we follow this test section by input x_2 at port 2 and the corresponding transition t takes the FSM to a state s_5 and produces output $(y_1, y_2, -)$. Then $ports(t) = \{1, 2\}$ and so normally one would expect to be able to apply input at either port 1 or port 2 after t. However, at this point there are two possible observations at port 1: either $y_1y_1y_1$ or $y_1y_1y_1y_1$, depending on which path from s_1 to s_4 was followed. In addition, one of these ($y_1y_1y_1$) is an observation that might have been made in state s_4. Thus, the tester at port 1 need not be able to determine when s_5 has been reached if t follows the test section from s_1 to s_4.

Let us suppose that $(s, \overline{\overline{x}}, s', \mathcal{P}, \mathcal{P}') \in Reach(M)$ is used to reach s'. The example above shows that the restriction, on ports where one can apply inputs, may still be required *after* we apply an additional input x at $p \in \mathcal{P}$: even if the tester at p' observes output in response to x, the tester need not be able to know that the output was in response to x. This is because there may have been several possible observations at p' in response to a test section previously used. Naturally, there is no problem if the test section led to a fixed output sequence at port p; this is why we use $\mathcal{P}'$ in addition to $\mathcal{P}$ in tuples in $Reach(M)$ (see Definition 11). Thus, if we use $(s, \overline{\overline{x}}, s', \mathcal{P}, \mathcal{P}')$ then we impose the restriction that (in the current test sequence) no future input is applied at a port outside of $\mathcal{P}'$. We will achieve this by adding a second set of ports to the label of a vertex. A vertex with label $v^{\mathcal{P}_1,\mathcal{P}_2}$ will denote the situation in which (in controllable testing) input can be applied next at any port in $\mathcal{P}_1$ and in the current test sequence we require that no further input is applied at ports outside of $\mathcal{P}_2$. The graphs we construct will have that if $v^{\mathcal{P}_1,\mathcal{P}_2}$ is a vertex then $\mathcal{P}_1 \subseteq \mathcal{P}_2$. Similar to before, we will say that $(\bar{x}_1, \ldots, \bar{x}_m)$ can be applied from $v_s^{\mathcal{P}_1,\mathcal{P}_2}$ if for all $p \in P$ we have that $\bar{x}_p \neq \epsilon$ implies that $p \in \mathcal{P}_1$.

The second factor is that the addition of a new vertex $v_s^{\mathcal{P}_1,\mathcal{P}_2}$ is only useful if this provides potential for test execution that is not provided by current vertices; if it is not subsumed by the current vertices.

Algorithm 1 $Update(G, R)$: Updating graph G

Input $G = (V, E)$ and $R \subseteq Reach(M)$
$V' = V$
while $V' \neq \emptyset$ **do**
  Choose some $v_s^{\mathcal{P}_1,\mathcal{P}_2} \in V'$
  $V' = V' \setminus \{v_s^{\mathcal{P}_1,\mathcal{P}_2}\}$
  for all $r = (s, (\bar{x}_1, \ldots, \bar{x}_m), s', \mathcal{P}, \mathcal{P}') \in R$ **do**
    $\mathcal{P}'_1 = \mathcal{P} \cap \mathcal{P}_2$
    $\mathcal{P}'_2 = \mathcal{P}' \cap \mathcal{P}_2$
    if $\forall 1 \leq p \leq m : (\bar{x}_p \neq \epsilon \Rightarrow p \in \mathcal{P}_1)$ **and** $v_{s'}^{\mathcal{P}'_1,\mathcal{P}'_2}$ is not subsumed by V **then**
      $V = V \cup \{v_{s'}^{\mathcal{P}'_1,\mathcal{P}'_2}\}$
      $V' = V' \cup \{v_{s'}^{\mathcal{P}'_1,\mathcal{P}'_2}\}$
      $E = E \cup \{(v_s^{\mathcal{P}_1,\mathcal{P}_2}, (\bar{x}_1, \ldots, \bar{x}_m), v_{s'}^{\mathcal{P}'_1,\mathcal{P}'_2})\}$
    end if
  end for
end while
Output (V, E)

Definition 12. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM with port set P and let us consider a graph $G = (V, E)$. Given a state $s \in S$ and sets $\mathcal{P}_1, \mathcal{P}_2 \subseteq P$, we say that a vertex $v_s^{\mathcal{P}_1,\mathcal{P}_2}$ is* subsumed *by the set V of vertices if for all $p \in \mathcal{P}_1$ there exist $\mathcal{P}'_1, \mathcal{P}'_2$ such that $v_s^{\mathcal{P}'_1,\mathcal{P}'_2} \in V$ and $p \in \mathcal{P}'_1$.*

This definition ignores $\mathcal{P}_2$; we do this in order to limit the size of the extended graph we form (we avoid a, potentially exponential, subset construction). The factors discussed above lead to the Update function in Algorithm 1 that extends the current graph G, whose vertices are of the form $v_s^{\mathcal{P}_1,\mathcal{P}_2}$, on the basis of a set $R \subseteq Reach(M)$. Having used Algorithm 1, there may now be potential to add new edges and further vertices that correspond to controllable testing from the vertices added. This process is outlined in Algorithm 2.

The overall algorithm starts with the traditional graph $G(M)$ as defined in Definition 7 and repeatedly applies the Update and Complete functions until a fixed point is found. This process is outlined in Algorithm 3 in which $G'(M)$ is the graph G in which a vertex of the form $v_s^{\mathcal{P}}$ is renamed $v_s^{\mathcal{P},P}$.

Example 3. Consider the (part of an) FSM M in Figure 2. $G(M)$ is showed in Figure 4 (non-dotted vertices and lines). Next we explain how Algorithm 3 works. Consider test section (ϵ, x_2, x_3) and $R = \{(s_1, (\epsilon, x_2, x_3), s_4, \{2, 3\}, \{2, 3\})\}$. Note that ports 2 and 3 are termination aware, because the conditions included in Definition 10 are satisfied.

The application of the update function to $G(M)$ and R creates a new vertex $V_{s_4}^{\{2,3\},\{2,3\}}$ and a new edge $(V_{s_1}^{\{1,2,3\}\{1,2,3\}}, (-, x_2, x_3), V_{s_4}^{\{2,3\},\{2,3\}})$ in the graph (see the dotted edge and vertex).

Next we consider the complexity of constructing the final graph, assuming that R is given. A vertex $v_s^{\mathcal{P}_1,\mathcal{P}_2}$ is added if it is not subsumed by the current

Algorithm 2 *Complete*(G, M): Completing graph G

Input $G = (V, E)$ and FSM M
$V' = V$
while $V' \neq \emptyset$ **do**
  Choose some $v_s^{\mathcal{P}_1,\mathcal{P}_2} \in V'$
  $V' = V' \setminus \{v_s^{\mathcal{P}_1,\mathcal{P}_2}\}$
  for all $p \in \mathcal{P}_1$ **and** $x \in X_p$ **do**
    $s' = \delta(s, x)$ and $y = \lambda(s, x)$
    $\mathcal{P}' = ports(x/y)$, $\mathcal{P}'_1 = \mathcal{P}' \cap \mathcal{P}_2$
    if $v_{s'}^{\mathcal{P}'_1,\mathcal{P}_2}$ is not subsumed by a vertex in V **then**
      $V = V \cup \{v_{s'}^{\mathcal{P}'_1,\mathcal{P}_2}\}$
      $V' = V' \cup \{v_{s'}^{\mathcal{P}'_1,\mathcal{P}_2}\}$
      $E = E \cup \{(v_s^{\mathcal{P}_1,\mathcal{P}_2}, x, v_{s'}^{\mathcal{P}'_1,\mathcal{P}_2})\}$
    end if
  end for
end while
Output (V, E)

Algorithm 3 Generating the graph G

Input FSM M, $G(M) = (V, E)$ and $R \subseteq Reach(M)$
$V' = \{v_s^{\mathcal{P},\mathcal{P}} | v_s^{\mathcal{P}} \in V\}$
$E' = \{(v_s^{\mathcal{P},\mathcal{P}}, a, v_{s'}^{\mathcal{P}',\mathcal{P}'}) | (v_s^{\mathcal{P}}, a, v_{s'}^{\mathcal{P}'}) \in E\}$
$G' = (V', E')$
repeat
  $G = G'$
  $G' = Complete(Update(G, R), M)$
until $G = G'$
Output G

vertices and this is the case if and only if there exists $p \in \mathcal{P}'$ such that no current vertex $v_s^{\mathcal{P}'_1,\mathcal{P}'_2}$ has $p \in \mathcal{P}'_1$. If $v_s^{\mathcal{P}_1,\mathcal{P}_2}$ is added then this increases the number of ports p such that there is a vertex $v_s^{\mathcal{P}'_1,\mathcal{P}'_2}$ with $p \in \mathcal{P}'_1$. As a result, given state s, Algorithm 3 can add at most m vertices of the form $v_s^{\mathcal{P}_1,\mathcal{P}_2}$. Therefore, if R has already been produced then Algorithm 3 is a polynomial time algorithm. In the next section we explore the case where there are bounds on test section size and the complexity of the problem of generating $Reach(M)$ in this situation.

4 Bounding convergent test sections

In the previous section we showed how $G(M)$ can be extended using test sections. In principle such test sections might be arbitrarily long but we will want to use relatively short test sections if we want testing to be efficient. Thus, in practice one might want to place upper bounds on the lengths of test sections used. The

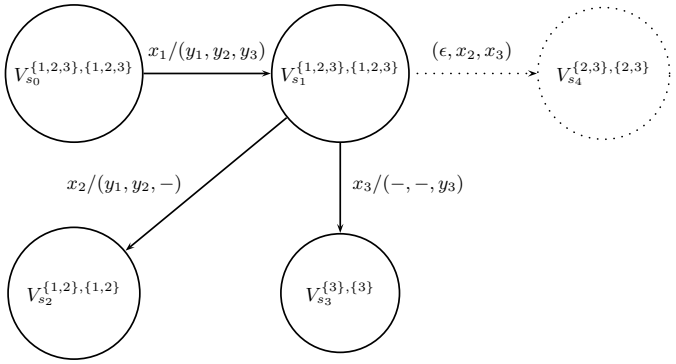


Fig. 4: Extension of $G(M)$

following two results provide additional motivation; they show that even the process of checking whether a test section is convergent is coNP-complete.

Theorem 1. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM, $s, s' \in S$ be states of M, and $\overline{\overline{x}}$ be a test section. The problem of deciding whether $\overline{\overline{x}}$ is convergent from s to s' is coNP-complete.*

Proof. We start by proving that the problem is in coNP. A non-deterministic Turing machine might guess an interleaving $\bar{x} \in INT(\overline{\overline{x}})$ and check whether $\bar{x}$ takes M from s to s'. Since this process takes polynomial time, a non-deterministic Turing machine can decide in polynomial time whether there is an interleaving of $\overline{\overline{x}}$ take M from s to a state other than s'. Thus, the problem of deciding whether all interleavings of $\overline{\overline{x}}$ take M from s to s' is in coNP.

We now prove that the problem is coNP-hard by relating it to the negation of the (NP-complete) Hamiltonian Path Problem (HPP). Let us suppose that we are given a directed graph G and we wish to solve the HPP, which is to determine whether there is a path that includes all vertices exactly once. Let $v_1, \ldots, v_n$ be the vertices of G. We will construct an FSM M with inputs $x_1, \ldots, x_n$ (each x_i at a separate port i) and states $s_0, s_1, \ldots, s_n, s_e$ as follows.

1. From the initial state s_0 input x_i takes M to s_i.
2. In state $s_i \neq s_e$, input x_j has the following effect:
   (a) If G contains an edge from v_i to v_j then we include a transition to s_j.
   (b) Otherwise there is a transition to the "error state" s_e.
   (c) In state s_e, all inputs lead to no change of state.
   (d) The outputs of the transitions can be chosen arbitrarily.

Now consider the test section $(x_1, \ldots, x_n)$ and its possible interleavings. The key observation is that an interleaving $\bar{x} = x_{i_1} \ldots x_{i_n}$ of $(x_1, \ldots, x_n)$ takes M from s_0 to a state other than s_e if and only if $v_{i_1} \ldots v_{i_n}$ is a path of G and such a path of G must be a Hamiltonian path of G. Thus, G has a Hamiltonian path

if and only if $(x_1, \ldots, x_n)$ is not convergent from s_0 to s_e for M. Since the HPP is NP-hard, this means that it is NP-hard to check that a test section is not convergent. The result therefore follows.

Theorem 2. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM, $s \in S$ be a state of M, and $\overline{\overline{x}}$ be a test section. The problem of deciding whether $\overline{\overline{x}}$ is convergent from s is coNP-complete.*

Proof. The problem being in coNP follows from Theorem 1 and there being polynomially many states. The proof that the problem is coNP-hard follows in the same way as the proof of Theorem 1 since in the constructed FSM we have that the test section $(x_1, \ldots, x_n)$ is convergent if and only if it converges to s_e (if the final state is not s_e, for interleaving $x_{i_1} \ldots x_{i_n}$, then it is s_{i_n}).

Since we want to have efficient algorithms, we now explore the case where we place an upper bound on the size of test sections considered.

Definition 13. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM, $s, s' \in S$ be states of M, $k \geq 0$, and $\overline{\overline{x}} = (\bar{x}_1, \ldots, \bar{x}_m)$ be a test section. $\overline{\overline{x}}$ is* k-convergent *from s to s' if $\overline{\overline{x}}$ is* convergent *from s to s' and $\sum_{p=1}^{m} |\bar{x}_p| \leq k$. Further, $\overline{\overline{x}}$ is* k-convergent *from s if there exists a state s' such that $\overline{\overline{x}}$ is* k-convergent *from s to s'.*

Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM and $k \geq 0$. We define $Reach(M, k)$ as the following subset of $Reach(M)$

$$\left\{ (s, (\bar{x}_1, \ldots, \bar{x}_m), s', \mathcal{P}, \mathcal{P}') \in Reach(M) \left| \sum_{p=1}^{m} |\bar{x}_p| \leq k \right. \right\}$$

Importantly, if we place an upper bound on k, or we fix k, then the number of interleavings defined by a test section is also bounded. As a result, the process of checking which states are reached using interleavings of a test section takes polynomial time. We therefore obtain the following results.

Theorem 3. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM, $s \in S$ be a state, $k \geq 0$, and $\overline{\overline{x}}$ be a test section. If k is bounded then*

1. *Given state $s' \in S$, the problem of deciding whether $\overline{\overline{x}}$ is k-convergent from s to s' can be decided in polynomial time.*
2. *The problem of deciding whether $\overline{\overline{x}}$ is k-convergent from s can be decided in polynomial time.*

The next result follows from the fact that for bounded k the number of possible test sections, and the number of interleavings of each one, are bounded by polynomials.

Theorem 4. *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be an FSM and $k \geq 0$. If k is bounded then the problem of generating $Reach(M, k)$ has polynomial time complexity.*

This shows that if we bound (or fix) k then we can compute $Reach(M, k)$ in polynomial time and so Algorithm 3 takes polynomial time. On the contrary, from Theorem 1, we have that this result does not hold if we do not bound k (unless $P = NP$). This suggests that Algorithm 3 can be applied with the entire set $Reach(M, k)$ if one wishes to restrict attention to a relatively small value of k but otherwise one might use heuristics to generate some $R \subseteq Reach(M)$.

5 Conclusions

This paper concerned testing in the distributed test architecture, where a local tester only observes events at its port, the testers do not synchronise, and there is no global clock. Almost all test generation algorithms, for testing from an FSM in the distributed test architecture, return controllable test sequences but this can be restrictive. For example, an FSM specification M may have states that cannot be reached in controllable testing. We introduced the notion of a test section, which contains a fixed input sequence for each port. We showed how test sections can be used to weaken the classical notion of controllability: rather than require that the path of the FSM specification M traversed is uniquely determined, we instead require that there is only one state of M that can be reached by a test section (the test section is convergent). Thus, the notion of a test section being convergent is similar to partial order reduction. We showed how, given a set R of convergent test sections, one can derive a directed graph G that describes what can be achieved using these test sections. In general, one cannot expect to generate all convergent test sections, since this set might be infinite. However, we found that if one bounds the size of the test sections then one can generate the complete set (that satisfies this upper bound) in polynomial time. As a result, one can also generate the graph G in polynomial time.

There are several possible lines of future work. First, it would be interesting to explore alternative conditions under which one can efficiently generate the set $Reach(M)$. There is also the potential for the approach to be generalised to allow test sections whose components are adaptive (the next input depends on the observed output) and also to non-deterministic FSMs. One might also explore notions of coverage. Finally, one might implement the proposed technique in a tool and then carry out industrial case studies.

References

1. G. von Bochmann, S. Haar, C. Jard, and G.-V. Jourdan. Testing systems specified as partial order input/output automata. In *Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047*, pages 169–183. Springer, 2008.
2. L. Cacciari and O. Rafiq. Controllability and observability in distributed testing. *Information and Software Technology*, 41(11–12):767–780, 1999.
3. T. S. Chow. Testing software design modeled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178–187, 1978.

4. 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.
5. R. Dssouli and G. von Bochmann. Conformance testing with multiple observers. In *6th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'86*, pages 217–229. North-Holland, 1986.
6. P. Godefroid. Using partial orders to improve automatic verification methods. In *2nd Int. Workshop on Computer Aided Verification, CAV'90, LNCS 531*, pages 176–185. Springer, 1991.
7. S. Haar, C. Jard, and G.-V. Jourdan. Testing input/output partial order automata. In *Joint 19th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'07, and 7th Int. Workshop on Formal Approaches to Software Testing, FATES'07, LNCS 4581*, pages 171–185. Springer, 2007.
8. R. M. Hierons. Canonical finite state machines for distributed systems. *Theoretical Computer Science*, 411(2):566–580, 2010.
9. R. M. Hierons. Reaching and distinguishing states of distributed systems. *SIAM Journal on Computing*, 39(8):3480–3500, 2010.
10. R. M. Hierons. Generating complete controllable test suites for distributed testing. *IEEE Transactions on Software Engineering*, 41(3):279–293, 2015.
11. 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.
12. R. M. Hierons and H. Ural. UIO sequence based checking sequences for distributed test architectures. *Information and Software Technology*, 45(12):793–803, 2003.
13. R. M. Hierons and H. Ural. Checking sequences for distributed test architectures. *Distributed Computing*, 21(3):223–238, 2008.
14. Joint Technical Committee ISO/IEC JTC 1. *International Standard ISO/IEC 9646-1. Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts*. ISO/IEC, 1994.
15. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
16. 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.
17. E. P. Moore. Gedanken experiments on sequential machines. In C. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.
18. H. Ponce de León, S. Haar, and D. Longuet. Unfolding-based test selection for concurrent conformance. In *25th IFIP WG 6.1 Int. Conf. on Testing Software and Systems, ICTSS'13, LNCS 8254*, pages 98–113. Springer, 2013.
19. H. Ponce de León, S. Haar, and D. Longuet. Model-based testing for concurrent systems: unfolding-based test selection. *International Journal on Software Tools for Technology Transfer*, 18(3):305–318, 2016.
20. O. Rafiq and L. Cacciari. Coordination algorithm for distributed testing. *The Journal of Supercomputing*, 24(2):203–211, 2003.
21. B. Sarikaya and G. von Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, 32:389–395, 1984.
22. H. Ural and D. Whittier. Distributed testing without encountering controllability and observability problems. *Information Processing Letters*, 88(3):133–141, 2003.
23. H. Ural and C. Williams. Constructing checking sequences for distributed testing. *Formal Aspects of Computing*, 18(1):84–101, 2006.