Noname manuscript No.
(will be inserted by the editor)

R. M. Hierons · H. Ural

Checking Sequences for Distributed Test Architectures

the date of receipt and acceptance should be inserted later

Abstract Controllability and observability problems may manifest themselves during the application of a checking sequence in a test architecture where there are multiple remote testers. These problems often require the use of external coordination message exchanges among testers during testing. However, the use of coordination messages requires the existence of an external network that can increase the cost of testing and can be difficult to implement. In addition, the use of coordination messages introduces delays and this can cause problems where there are timing constraints. Thus, sometimes it is desired to construct a checking sequence from the specification of the system under test that will be free from controllability and observability problems without requiring the use of external coordination message exchanges. This paper gives conditions under which it is possible to produce such a checking sequence, using multiple distinguishing sequences, and an algorithm that achieves this.

Keywords Testing · Checking sequence · distributed test architecture · coordination problems · observability problems

R. M. Hierons
School of Information Systems, Computing and Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
E-mail: rob.hierons@brunel.ac.uk

H. Ural
School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, K1N 6N5, Canada E-mail: ural@site.uottawa.ca

1 Introduction

The importance and high cost of software testing has led to much interest in automated test generation. Among various testing activities in software development that benefited most from automated test generation is model based testing [1,13,16] where a model of the software under test is used for generating tests. A particular area of application of model based testing is system level testing of reactive systems where the required externally observable behavior of the system under test (SUT) is modeled by a syntactically finite representation of all possible valid sequences of interactions of the system components with their external environment. Within the context of testing state-based systems, the externally observable behavior of the SUT is typically expressed in terms of a Finite State Machine (FSM) M.

Then the system testing of the SUT is carried out by applying a test sequence, that has been generated from M, at its interfaces with its environment. In some cases it is possible to produce a *checking sequence*: a test sequence that is guaranteed to determine whether the SUT behaves as specified in the FSM M representing its desired behavior [14,15,19,23,44]. A test or checking sequence is applied within a given test architecture and the resulting output sequence is checked against the FSM M.

A multi-port FSM can be used to express the expected externally observable behavior of potential implementations of a distributed system which can have multiple interfaces, called ports. In a multi-port FSM, each transition is labelled with an input from a port and an output vector consisting of a (possibly empty) output to each port. In system testing of a distributed system N, a *distributed test architecture* can be used where a tester is placed at each port of the SUT N, the testers cannot communicate with one another and there is no global clock.

During the application of a checking sequence to N in a distributed test architecture, the use of multiple testers introduces the possibility of coordination problems amongst remote testers (see, for example, [2,4,5,8,11,12,17,24,36,37,41–43,45,47]). These potential problems are known as *controllability* and *observability* problems. These problems occur if a tester cannot determine either when to apply a particular input to N, or whether a particular output from N was generated in response to a specific input, respectively. The controllability (synchronization) problem occurs when the tester at a port p is expected to send an input to N after N responds to an input from the tester at some $q \neq p$, without sending an output to p. For example, consider a distributed test architecture in which there are remote testers at two ports U and L. If the input of x at port U is expected to lead to output y at U only and this is to be followed by input x' at L then the tester at L does not know when to send x' since it did not observe either x or y. The observability problem occurs when the tester at some port p is expected to receive an output from N in response to a given input and is unable to determine when to start and stop waiting. Observability problems hamper the detectability of *output-shifting faults* in N i.e., an output associated with the current input is generated by N in response to either some earlier input or some later input. Let us suppose, for example, that in testing the input of x at U is expected to lead to the output of y_U at U and y_L at L, this is to be followed by input x' at U, and this should result in the output of y'_U at U. Then, the expected sequences of observations are seen by each tester if instead the input of x leads to output of y_U at U and then the input of x' leads to output y'_U at U and y_L at L: the tester at U sees $xy_Ux'y'_U$ and the tester at L sees y_L.

The use of the distributed test architecture can lead to controllability and observability problems and so can make test generation complex and reduce test effectiveness. However, if the interfaces are physically distributed then the alternative is to connect the testers through an external network. The deployment of such a network can make testing more expensive and the delays introduced by the exchange of external coordination messages between testers can make testing take longer. In addition, the exchange of such messages between testers can lead to delays that mean that some tests with timing constraints cannot be implemented. For example, let us suppose that we wish to follow input x at port p_i by input x' at port $p_j \neq p_i$ and this is to be achieved by an external coordination message being sent from the tester at p_i to the tester at p_j after the input of x. If the external coordination messages take time t to arrive and the input of x' must occur within time t' of x with $t' < t$ then this approach will not work. The timing issues can be particularly problematic if the SUT responds rapidly relative to the network used for external coordination messages. See [31] for a discussion of some timing issues that arise when using external coordination messages. Naturally, if we have access to the source code of the SUT, and potentially can change this, there are other ways of overcoming these problems.

This paper considers the problem of testing from an FSM in the distributed test architecture where the focus is system level testing. This problem has largely been studied in the context of protocol conformance testing. However, it is potentially relevant whenever testing a deterministic state-based system that has physically distributed interfaces. If the system is implemented through a set of state-based subsystems that interact, then there is the potential to combine the FSM models of these subsystems to form a single FSM for system level testing. However, if the focus of testing is unit level or integration testing then Communicating FSM (CFSM) based models can be employed to facilitate automated test generation where interactions among the subsystems are taken into consideration [9]. Naturally, distributed systems are often nondeterministic and it would thus be interesting to extend the work to the problem of testing from a nondeterministic FSM in this architecture. However, there is the potential to adapt approaches, such as the one given in this paper, to testing from a deterministic FSM by using *deterministic testing*: test methods that make a nondeterministic distributed system behave in a deterministic manner during testing by forcing a given sequence of interleavings to occur (see, for example, [18, 26,34,35]).

This paper makes the following contributions. It gives a method for constructing checking sequences from multiport FSMs that can be applied in a distributed test architecture without encountering controllability and observability problems and without using external coordination messages among testers. First we show how a checking sequence can be produced where there are controllability problems but not observability problems. This is the case, for example, when a global clock can be used to timestamp the inputs and outputs and it is guaranteed that all of the outputs produced in response to an input are observed before the next input. We then show how this can be extended to a checking sequence in the case where there can be observability problems. Naturally, since such checking sequences do not always exist, the algorithms work under certain stated assumptions.

This is the first paper that shows how such checking sequences can be produced without using a reliable reset operation[1]. In this paper we rely on the existence of distinguishing sequences[2] for state verification rather than alternatives such as unique input/output sequences or a characterization set. This choice was made because even for single-port FSMs there is no known method for generating a polynomial size checking sequence using these alternative approaches for state verification. Note that some recent work has investigated the problem of checking the output of transitions while avoiding controllability and observability problems but this previous work assumes that each transition of the SUT has the correct final state [5–7].

The rest of the paper is organized as follows: Section 2 introduces the terminology used in this paper. Section 3 defines a property, of a set $\mathcal{D}$ of distinguishing sequences, that must hold in order for us to be able to use $\mathcal{D}$ to check the final state of each transition of the multi-port FSM M. Section 4 then gives an algorithm for generating a checking sequence that has no controllability problems. Section 5 introduces additional conditions and shows how, under these conditions, we can produce a checking sequence even if there can be observability problems. Finally Section 6 gives the concluding remarks.

[1] A reliable reset is a function that is guaranteed to take the implementation back to its initial state irrespective of its current state. The SUT need not have a reliable reset and even when it does the inclusion of resets can reduce test effectiveness and may require human involvement and thus greatly increase the cost of test execution [3, 20, 46]

[2] Given an FSM M, an input sequence is a distinguishing sequence for M if it leads to n different output sequence from the n different states of M. Distinguishing sequences are formally defined in Section 2.

2 Preliminaries

2.1 Multi-port FSMs

A (deterministic) multi-port FSM M has $m > 1$ ports at which it interacts with its environment. The m ports of M are identified by integers in the set $[1, m] = \{1, \ldots, m\}$. A multi-port FSM with m ports is defined by a tuple $(S, X, Y, \delta, \lambda, s_1)$ in which:

- S is the finite *set of states* of M;
- $s_1 \in S$ is the *initial state* of M;

– $X = \bigcup_{i=1}^{m} X_i$ is the finite *input alphabet* of M, where X_i is the input alphabet of port i, $X_i \cap X_j = \emptyset$ for all $i, j \in [1, m]$, $i \neq j$;
– $Y = \prod_{i=1}^{m}(Y_i \cup \{-\})$ is the finite *output alphabet* of M, where Y_i is the output alphabet of port i, $Y_i \cap Y_j = \emptyset$ for all $i, j \in [1, m]$, $i \neq j$, and $-$ means null output;
– δ is the *transition function* of type $S \times X \rightarrow S$; and
– λ is the *output function* of type $S \times X \rightarrow Y$.

This paper deals with multi-port FSMs and so a multi-port FSM is called an FSM; an FSM with only one port is called a *single-port FSM.* M denotes a multi-port FSM that models the expected behaviour of the SUT $N = (U, X, Y, \delta_N, \lambda_N, u_1)$. A variable name has a bar over it (for example, $\bar{x}$) if this variable represents a sequence and ϵ denotes the empty sequence. A sequence can be represented by listing its elements. For example *abc* represents the sequence with three elements: a then b and then c.

Note that each $y \in Y$ is a *vector of outputs*, i.e., $y = \langle o_1, o_2, \ldots, o_m\rangle$ where $o_i \in Y_i \cup \{-\}$ for $i \in [1, m]$. In the following, $p \in [1, m]$ is a port, $x \in X$ is a general input, and the label $y[p, o]$ is used to denote an output vector y of a transition where the output at port p is o. We use $y\mid_p$ to denote the output at port p in y. It is possible to extend δ and λ to input sequences in the following way: $\delta(s, \epsilon) = s$, $\delta(s, x\bar{x}) = \delta(\delta(s, x), \bar{x})$, $\lambda(s, \epsilon) = \epsilon$, and $\lambda(s, x\bar{x}) = \lambda(s, x)\lambda(\delta(s, x), \bar{x})$. A transition τ of an FSM M is a triple $(s_j, s_k, x/y)$, where $s_j, s_k \in S$, $x \in X$, and $y \in Y$ such that $\delta(s_j, x) = s_k$, $\lambda(s_j, x) = y$, and s_j and s_k are the *starting state* and the *ending state* of τ, respectively. The *input/output pair* x/y is the *label* of τ.

A *path* $\bar{\rho} = \tau_1\,\tau_2\,\ldots\,\tau_k$ $(k \geq 0)$ is a finite sequence of transitions such that if $k \geq 2$ then the ending state of τ_i is the starting state of τ_{i+1} for all $i \in [1, k-1]$. Path $\bar{\rho}$ is said to *start* at the starting state of τ_1. When the ending state of the last transition of path $\bar{\rho}_1$ is the starting state of the first transition of path $\bar{\rho}_2$, we use $\bar{\rho}_1\bar{\rho}_2$ to denote the *concatenation* of paths $\bar{\rho}_1$ and $\bar{\rho}_2$. The *label* of a path $\bar{\rho} = (s_1, s_2, x_1/y_1)\,(s_2, s_3, x_2/y_2)\ldots(s_k, s_{k+1}, x_k/y_k)$ $(k \geq 1)$ is the sequence of input/output pairs $x_1/y_1x_2/y_2\,\ldots\,x_k/y_k$, which is called an *input/output sequence.* At times we will want to reason about the state of the SUT after a prefix of an input/output sequence and in order to assist with this we will consider an input/output sequence $x_1/y_1x_2/y_2\,\ldots\,x_k/y_k$ to be a sequence of *edges* $(n_1, n_2, x_1/y_1)\ldots(n_k, n_{k+1}, x_k/y_k)$ in which $n_1, \ldots, n_{k+1}$ are called *nodes.* The *input portion* and *output portion* of an input/output sequence $x_1/y_1x_2/y_2\,\ldots\,x_k/y_k$ are the input sequence $x_1x_2\,\ldots\,x_k$ and output sequence $y_1y_2\,\ldots\,y_k$, respectively. The input sequence $x_1\ldots x_k$ (or input/output sequence $x_1/y_1x_2/y_2\ldots x_k/y_k$) is said to *label* $\bar{\rho}$. Note that we call a sequence of input/output pairs $x_1/y_1x_2/y_2\,\ldots\,x_k/y_k$, or $\bar{x}/\bar{y}$ ($\bar{x} = x_1\ldots x_k$ and $\bar{y} = y_1\ldots y_k$) or any combination of these an input/output sequence.

An FSM M is said to be *globally minimal*[3] if none of its states are *globally equivalent*[4] (i.e., for all $s_i, s_j \in S$, $s_i \neq s_j$, there exists an input sequence $\bar{x} \in X^*$ such that $\lambda(s_i, \bar{x}) \neq \lambda(s_j, \bar{x})$). Any FSM can be converted into an equivalent globally minimal FSM: this process is equivalent to the minimization of a single-port FSM and for an n state FSM with p inputs this can be achieved in time of $O(pn \log n)$ [25]. Throughout this paper we thus assume that any FSM considered is globally minimal.

In order to reason about test effectiveness it is normal to use a fault model: a set Φ_M of FSMs such that we believe that the SUT behaves like an unknown element of Φ_M [29]. The purpose of the fault model is to capture the types of faults that it is believed can occur and to therefore make it possible to reason about test effectiveness. We use a standard fault model Φ_M from protocol conformance testing, which is the set of FSMs that have the same input and output alphabets as M and no more states than M. Input sequence $\bar{x}$ is a *checking sequence* if $\bar{x}$ distinguishes M from every element of Φ_M that is not equivalent to M. In this paper we are concerned with the problem of generating a checking sequence and thus obtaining a sequence that provides *full fault coverage* with respect to the fault model. We will see that the notions of equivalence of SUT $N \in \Phi_M$ and M and distinguishing N from M depends upon the test architecture used and whether there can be observability problems.

There are several benefits to producing a checking sequence rather than a test sequence that, for example, includes subsequences that aim to check each transition of M. First, we know that if the SUT passes a checking sequence then either it is correct or our initial assumption was incorrect: the SUT is not equivalent to a member of the fault model. This gives information regarding the types of faults that can be missed and provides some guarantees. Second, there is experimental evidence that checking sequences are more effective in distinguishing between an FSM M and faulty FSMs [10]. Naturally, there is scope for using a larger fault model, such as the set of FSMs with the same input and output alphabets as M and at most δ extra states for some predetermined δ as it was shown in [33] but the use of such fault models in the problem studied in this paper is a topic for future work. It would also be interesting to extend this test method to the case where the externally observable behavior of the system is modelled as a nondeterministic FSM.

2.2 The distributed test architecture

An FSM M defines the set of expected global behaviours of any potential implementation. Each expected global behaviour is expressed as the label of a sequence of tran-

[3] Normally such an FSM is said to be minimal. In this paper the phrase globally minimal is used in order to distinguish this from the notion of local minimality described in Section 2.4.

[4] The term globally equivalent is used, rather than equivalent, in order to distinguish this from the term locally equivalent described in Section 2.4.

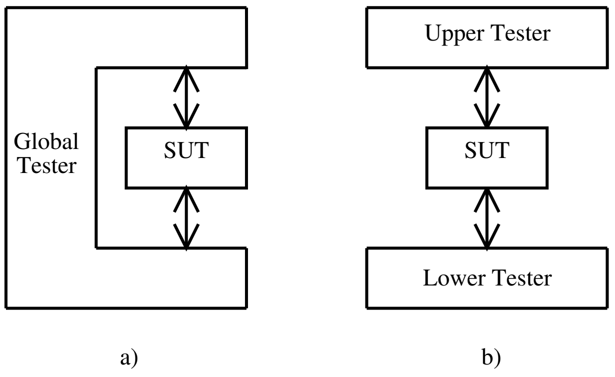

Fig. 1 Two test architectures: a) local; b) distributed

sitions from M. An expected global behaviour is called a global input/output sequence.

Testing SUT $N \in \Phi_M$ whose expected externally observable behaviour is defined by FSM M can be carried out as a fault detection experiment [14,19] in a specific test architecture. Two standardized architectures [28] are shown in Figure 1 for a two-port SUT. The two ports, U and L, represent the upper interface and lower interface of the SUT respectively. The local architecture in Figure 1a) has a global tester that controls and observes both interfaces of the SUT. Figure 1b) shows the distributed test architecture where the lower interface and the upper interface of the SUT are controlled and observed by separate testers. Each tester applies its own local view constructed from a global input/output sequence for the SUT. In the local view, a tester can't observe the inputs or outputs of the other testers.

In Figure 1b) there is no global tester. Instead, U and L are two remote testers that are required to coordinate the application of a global input/output sequence. However, they cannot directly communicate with one another and there may be no global clock. This requirement can lead to controllability and observability problems that are defined below.

2.3 Controllability (synchronization) and observability problems

Let us suppose that in testing input x at port U is expected to lead to output y_U at U only and this is to be followed by the input of x' at L. Then we have a controllability problem since the tester at L does not observe either the input or output from the previous transition and so does not know when to send input x' to the SUT. In general, given an FSM M and input/output sequence $x_1/y_1x_2/y_2 \ldots x_k/y_k$ of M, a *controllability (synchronization) problem* occurs when, in the labels x_i/y_i and x_{i+1}/y_{i+1} of two consecutive transitions, there exists a port $p \in [1, m]$ such that $x_{i+1} \in X_p$, $x_i \notin X_p$, and $y_i \mid_p = -$. If there is such a synchronization problem then we cannot apply x_{i+1} after x_i when testing in the distributed test architecture since the tester at port p cannot know when to apply x_{i+1}. Two consecutive transitions τ_i and τ_{i+1} whose labels are x_i/y_i and x_{i+1}/y_{i+1}, form a *synchronizable pair* of transitions if τ_{i+1} can follow τ_i without causing a synchronization problem. Any (sub)sequence of transitions in which every pair of consecutive transitions is synchronizable is called a *synchronizable transition (sub)sequence.* An input/output sequence is *synchronizable* if it is the label of a synchronizable transition sequence. An FSM may have properties that make it inherently untestable within the distributed test architecture without using external coordination messages. For example, there may be no synchronizable input/output sequence that is the label of a path that includes a particular transition τ, in which case we cannot test transition τ without introducing a controllability problem. We thus make the following assumption regarding M.

Assumption 1 *For every pair τ, τ' of transitions of M there is some synchronizable transition sequence $\bar{\rho} = \tau_1 \dots \tau_k$ of M in which $\tau_1 = \tau$ and $\tau_k = \tau'$.*

Given transitions τ and τ' there are low order polynomial algorithms for producing such a minimal length transition sequence based on a directed graph in which paths correspond to synchronizable transition sequences (see, for example, [17]).

In the distributed test architecture each tester sees only the behaviour at a single port. Suppose that a sequence of interactions has occurred. Then the tester at each port sees only a portion of this: the parts that involved that port. We call this the *actual local behaviour*. The tester compares this with the *expected local behaviour*. If $\bar{z}$ is an input/output sequence then we use $\pi_p(\bar{z})$ to denote the corresponding sequence of inputs and outputs at port p. The projection function π_p can be defined by the following in which $\bar{z}$ is an input/output sequence and x is an input.

$$\pi_p(\epsilon) = \epsilon$$

$$\pi_p((x/y[p, -])\bar{z}) = \pi_p(\bar{z}) \text{ if } x \notin X_p$$

$$\pi_p((x/y[p, -])\bar{z}) = x\pi_p(\bar{z}) \text{ if } x \in X_p$$

$$\pi_p((x/y[p, o])\bar{z}) = o\pi_p(\bar{z}) \text{ if } x \notin X_p$$

$$\pi_p((x/y[p, o])\bar{z}) = xo\pi_p(\bar{z}) \text{ if } x \in X_p$$

Given a sequence $\tau_1 \dots \tau_k$ of transitions, $\tau_i = (s_i, s_{i+1}, x_i/y_i)$, and port p, $\pi_p(\tau_1 \dots \tau_k)$ denotes the sequence $\pi_p(x_1/y_1 \dots x_k/y_k)$.

Suppose the distributed test architecture is being used in testing SUT $N \in \Phi_M$ against an FSM M where $m = 2$ and the ports of M are denoted U and L. Suppose also that xx' is to be input when M is in state s, $x, x' \in X_U$, x is expected to trigger output (y_U, y_L) and x' is expected to trigger output $(y'_U, -)$. Then $xy_Ux'y'_U$ should occur at U and y_L should be observed at L. This is the case if $(y_U, -)$ is produced in response to x and (y'_U, y_L) is produced in response to x'. Since each tester only sees the interactions at its port, neither tester can observe these output faults[5] in this subsequence — the two output faults mask one another. This situation is represented in Figure 2 in which the differences in the two sequences of interactions cannot be observed by either tester. Naturally, we want to use tests in which output faults cannot mask one another in this way. Note that if x' had been from X_L, this combination of faults would have been de-

[5] An output fault is a fault in which a transition produces the wrong output.

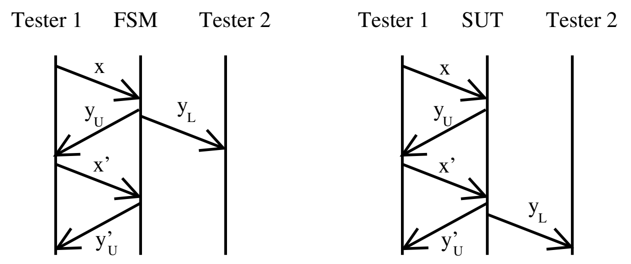

Fig. 2 Two behaviours that look equivalent to each tester

tected by the tester at L since $x'y_L$ would have occurred rather than y_Lx'.

The faults described above mask one another because the correct value y_L is observed at L, but due to the wrong transition, and the tester at port L does not know when to stop waiting for y_L. This corresponds to the notion of an undetectable forward output-shifting fault.

Definition 1 Let $\tau\bar{\rho}\tau'$ denote a synchronizable path with $\tau = (s_i, s_j, x/y)$ and $\tau' = (s_j, s_k, x'/y')$. Suppose also that for some $q \in [1, m]$ we have that $y\ |_q = o \neq -$, $y'\ |_q = -$, and no transition in $\bar{\rho}$ has output at q. Suppose there are faults in which the output at $p \in [1, m]$ is correct for τ and τ' (for all $p \neq q$) and at q the output in response to x is $-$ and the output at q in response to x' is o. This combination of faults is called a *forward output-shifting fault* [47]. It is an *undetectable forward output-shifting fault* if $x' \notin X_q$ and no transition from $\bar{\rho}$ has input at q; otherwise it is a *detectable forward output-shifting fault.*

A similar situation occurs if output at L is expected in response to x' but not x and instead it was produced in response to x.

Definition 2 Let $\tau\bar{\rho}\tau'$ denote a synchronizable path with $\tau = (s_i, s_j, x/y)$ and $\tau' = (s_j, s_k, x'/y')$ and for some $q \in [1, m]$ we have that $y\ |_q = -$, $y'\ |_q = o \neq -$, and no transition in $\bar{\rho}$ has output at q. Suppose there are faults in which the output at $p \in [1, m]$ is correct for τ and τ' (for all $p \neq q$) and at q the output in response to x is o and the output at q in response to x' is $-$. This combination of faults is called a *backward output-shifting fault* [47]. It is an *undetectable backward output-shifting fault* if $x' \notin X_q$ and no transition from $\bar{\rho}$ has input at q; otherwise it is a *detectable backward output-shifting fault.*

Definition 3 A fault is an *output-shifting fault* [36] if it is either a forward output-shifting fault or a backward output-shifting fault. An output-shifting fault is a *detectable output-shifting fault* if it is either a detectable forward output-shifting fault or a detectable backward output-shifting fault; otherwise it is an *undetectable output-shifting fault.*

Where output is shifted between two *adjacent* transitions, such output-shifting faults have been called *1-output-shifting faults* [2]. In this paper we consider the general case and not just 1-output-shifting faults.

The observability problem manifests itself in a checking sequence as an undetectable output-shifting fault. The following, which is proved in [22], relates the notion of an output-shifting fault being detectable to the definition of the projection π_p.

Proposition 1 *Given transitions τ and τ' of M such that $\tau\tau'$ is synchronizable, an output-shifting fault in $\tau\tau'$, which leads to the (faulty) transition sequence $\tau_1\tau_1'$ in SUT $N \in \Phi_M$, is a detectable output-shifting fault if and only if there is some port $p \in [1, m]$ such that $\pi_p(\tau\tau') \neq \pi_p(\tau_1\tau_1')$.*

2.4 Globally distinguishing and locally distinguishing states

This subsection, which is based on [22], defines what it means for an input sequence to distinguish two states in the distributed test architecture.

If there is a global tester, input sequence $\bar{x}$ distinguishes two states if the input of $\bar{x}$ leads to different output sequences when applied in these states. More formally, $\bar{x}$ *globally distinguishes* states s_i and s_j of M if and only if $\lambda(s_i, \bar{x}) \neq \lambda(s_j, \bar{x})$. This corresponds to the classical notion of distinguishing states of a single-port FSM. States s_i and s_j of M are *globally equivalent* if no input sequence globally distinguishes them and two FSMs are *globally equivalent* if their initial states are globally equivalent. Input sequence $\bar{x}$ is a *distinguishing sequence* for M if for every pair (s_i, s_j) of states of M, $\bar{x}$ globally distinguishes s_i and s_j.

In the distributed test architecture, if the testers can access a global clock then they could record the times at which inputs were applied and outputs observed. This would allow the reconstruction of the global input/output sequence if communication is synchronous or all outputs in response to an input are observed before the next input is sent (there is a *slow environment*). If the input/output sequence can be reconstructed then there are no observability problems and so global distinguishability applies.

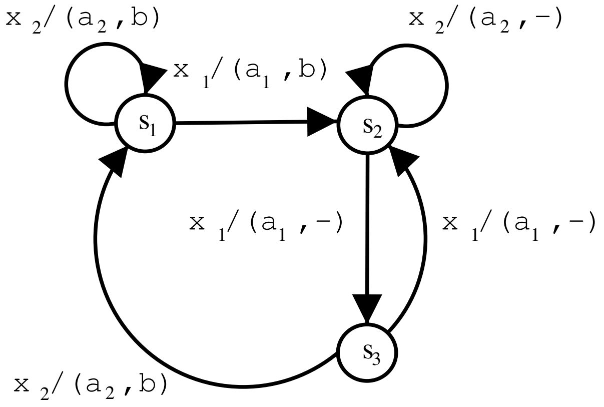

Fig. 3 The 2-port FSM M_0

Consider the FSM M_0 given in Figure 3 in which $x_1, x_2 \in X_U$, $a_1, a_2 \in Y_U$, and $b \in Y_L$. The input sequence x_1x_2 is a distinguishing sequence since it leads to three different output sequences: $(a_1, b)(a_2, -)$ from s_1, $(a_1, -)(a_2, b)$ from s_2, and $(a_1, -)(a_2, -)$ from s_3.

In M_0, x_1x_2 globally distinguishes states s_1 and s_2. However, neither tester observes a difference since for each state the expected local behaviour at L is b and the expected local behaviour at U is $x_1a_1x_2a_2$. In the distributed test architecture, if there is no global clock then x_1x_2 does not distinguish between s_1 and s_2 since it is necessary to observe some different sequence of input and output values at one of the ports: there is an observability problem. Given state s and input sequence $\bar{x}$, we use $\gamma(s, \bar{x})$ to denote the input/output sequence resulting from applying $\bar{x}$ when M is in state s. This can be recursively defined in the following manner: $\gamma(s, \epsilon) = \epsilon$; $\gamma(s, x\bar{x}) = (x/\lambda(s, x))\gamma(\delta(s, x), \bar{x})$. The function γ_N, for the SUT N, can similarly be defined. If input sequence $\bar{x}$ is applied when M is in state s_i the sequence $\pi_p(\gamma(s_i, \bar{x}))$ is observed at port p.

Definition 4 Input sequence $\bar{x}$ *locally distinguishes* states s_i and s_j of M *at port* $p \in [1, m]$ if $\bar{x}$ labels a synchronizable path from both s_i and s_j and $\pi_p(\gamma(s_i, \bar{x})) \neq \pi_p(\gamma(s_j, \bar{x}))$. Input sequence $\bar{x}$ *locally distinguishes* states s_i and s_j of M if there exists a port $p \in [1, m]$ such that $\bar{x}$ locally distinguishes s_i and s_j at p.

The following is proved in [22].

Proposition 2 *An input sequence may globally distinguish two states s_i and s_j but not locally distinguish them. Further, if $\bar{x} \in X^*$ locally distinguishes states s_i and s_j then $\bar{x}$ globally distinguishes s_i and s_j.*

Where there can be observability problems and external coordination messages are not used, in order to distinguish between states it is necessary to locally distinguish them. States s_i and s_j of M are *locally equivalent* if no input sequence locally distinguishes s_i and s_j. M is *locally minimal* if for every pair (s_i, s_j) of states of M, if $s_i \neq s_j$ then s_i and s_j are locally distinguishable.

It is possible to extend the notion of a distinguishing sequence to the distributed test architecture where there can be observability problems and external coordination messages are not used. Input sequence $\bar{x}$ is a *locally distinguishing sequence* for M if for all $s_i, s_j \in S$, $s_i \neq s_j$, $\bar{x}$ locally distinguishes s_i and s_j.

The problem of deciding whether an FSM has a distinguishing sequence is PSPACE-complete [32]. Thus, the problem of deciding whether an FSM has a locally distinguishing sequence is PSPACE-hard.

2.5 The proposed approach

Most checking sequence generation algorithms are based on a distinguishing sequence $\bar{D}$. Typically, they produce a set of subsequences and connect these subsequences in order to produce a checking sequence. Some of the subsequences check that $\bar{D}$ is a distinguishing sequence in the SUT and so $\bar{D}$ defines a bijection (one-to-one correspondence) between the states of the model and the states of the SUT. The *bijection* for a distinguishing sequence $\bar{D}$ is defined by: state s of M corresponds to state u of N if and only if the response of N to $\bar{D}$ when in state u is the same as the response of M to $\bar{D}$ when in state s. Other subsequences use $\bar{D}$ to check the transitions of the SUT. In order to check a transition $(s, s', x/y)$ we need to move to a state of the SUT that corresponds to s, apply input x, check that the SUT generates the output y and then apply $\bar{D}$ to check that the SUT is in the correct state after the transition.

This paper adapts this approach to the case where we are testing in the distributed test architecture. Again, the test generation algorithm produces a set of subsequences that can be connected to form a checking sequence. Since a transition t must be followed by input at a port that is involved in t, we may require a set $\{\bar{D}_1, \ldots, \bar{D}_r\}$ of distinguishing sequences rather than a single distinguishing sequence. Section 3 gives a sufficient condition under which $\bar{D}_1, \ldots, \bar{D}_r$ can be used to check the final state of every transition. It is thus not sufficient to check that each $\bar{D}_i$ is a distinguishing sequence in the SUT and thus defines a bijection between the states of the SUT and the states of the model: it is essential that the distinguishing sequences define the same bijection. In Section 4, Algorithm 1 shows how we can generate subsequences that check that a single distinguishing sequence $\bar{D}_1$ is also a distinguishing sequence in the SUT. Algorithm 2 shows how additional subsequence can be produced that use $\bar{D}_1$ to check that $\bar{D}_1, \ldots, \bar{D}_r$ are distinguishing sequences of the SUT that define the same bijection as $\bar{D}_1$. Algorithm 3 then shows how we can devise subsequences that check a transition using $\bar{D}_1, \ldots, \bar{D}_r$ and finally Algorithm 4 simply involves forming a single checking sequence from the subsequences returned by Algorithms 1, 2, and 3.

3 Using multiple distinguishing sequences

An input sequence $\bar{D}$ is a (globally) distinguishing sequence for M if it produces n different output sequences from the n different states of M. If these n different output sequences are seen in response to $\bar{D}$ in the SUT $N \in \Phi_M$ then since N has at most n states we know that $\bar{D}$ is also a distinguishing sequence for N. Where this is the case, $\bar{D}$ recognizes each state of N as a state of M. Since we are testing in the distributed test architecture we also require that for each state s of M the path from s with label $\bar{D}/\lambda(s, \bar{D})$ is synchronizable. The following adapts the definitions provided in [44] of what it means to recognize a node in a path $\bar{\rho}$ and to verify a transition of M in the label (input/output sequence) $\bar{Q}$ of $\bar{\rho}$. The base case is that the distinguishing sequence recognizes its starting state. The recursive cases essentially say that if an input sequence $\bar{x}$ is repeated in Q and in the two cases we know that the current state of the SUT must be the same before $\bar{x}$ is applied then the state of the SUT must be the same after these two occurrences of $\bar{x}$.

In order to prove that an input/output sequence $\bar{z}$ defines a checking sequence we will reason about the states of the SUT reached by prefixes of $\bar{z}$ and thus how the nodes visited by $\bar{z}$ correspond to states of M. This reasoning will be based on the use of distinguishing sequence $\bar{D}$ and the assumption that this defines a bijection between the states of the SUT and M: later we will see how we can produce subsequences with the property that if an SUT passes a test that contains these subsequences then $\bar{D}$ must define a bijection between the states of the SUT and M.

Definition 5 1. A node n_i of $\bar{\rho}$ is *d-recognized* in $\bar{Q}$ *by* $\bar{D}$ *as state* s of M if $\bar{D}/\lambda(s, \bar{D})$ is the label of a subpath of $\bar{Q}$ that starts at n_i. This says that, since we assume that $\bar{D}$ defines a bijection between the states of M and those of the SUT, if a node n_i is followed by a subpath labelled by $\bar{D}/\lambda(s, \bar{D})$ then n_i must correspond to state s.

2. Suppose that $(n_q, n_i, \bar{T})$ and $(n_j, n_k, \bar{T})$ are subpaths of $\bar{\rho}$ and $\bar{D}/\lambda(s, \bar{D})$ is a prefix of $\bar{T}$ (and thus n_q and n_j are d-recognized in $\bar{Q}$ by $\bar{D}$ as state s). Suppose also that node n_k is d-recognized by $\bar{D}$ as state s' of M. Then n_i is *t-recognized* in $\bar{Q}$ by $\bar{D}$ as s'. This says that if we know that two nodes n_q and n_j correspond to the same state, $\bar{T}$ labels a path from n_j to n_k and we know that n_k corresponds to state s then if there is a path with label $\bar{T}$ from n_q to n_i then, since the SUT is deterministic, n_i must correspond to state s.

3. Suppose that $(n_q, n_i, \bar{T})$ and $(n_j, n_k, \bar{T})$ are subpaths of $\bar{\rho}$ such that n_q and n_j are either d-recognized or t-recognized in $\bar{Q}$ by $\bar{D}$ as state s and n_k is either d-recognized or t-recognized in $\bar{Q}$ by $\bar{D}$ as state s'. Then n_i is *t-recognized* in $\bar{Q}$ by $\bar{D}$ as s'. This extends the previous case to allow the nodes n_q, n_j, and n_k to be t-recognized rather than being d-recognized.

4. If node n_i of $\bar{\rho}$ is either d-recognized or t-recognized in $\bar{Q}$ by $\bar{D}$ as state s then n_i is *recognized* in $\bar{Q}$ by $\bar{D}$ as state s. Where $\bar{D}$ is clear we say that n_i is *recognized* in $\bar{Q}$ as state s.

5. Transition $\tau = (s_a, s_b, x/y)$ of M is *verified* in $\bar{Q}$ by $\bar{D}$ if there is a subpath $(n_i, n_{i+1}, x_i/y_i)$ of $\bar{\rho}$ such that n_i is recognized in $\bar{Q}$ by $\bar{D}$ as s_a, n_{i+1} is recognized in $\bar{Q}$ by $\bar{D}$ as s_b, $x_i = x$ and $y_i = y$.

Given a transition τ of M, we use $P(\tau)$ to denote the set of ports that are involved in τ: the port that receives the input of τ and each port that receives non-empty output from τ. Transition τ can be *followed* by an input at port p, without causing a synchronization problem, if and only if $p \in P(\tau)$. Given an input sequence $\bar{D}$, *inport*$(\bar{D})$ denotes the port whose tester sends the first input from $\bar{D}$.

A distinguishing sequence $\bar{D}$ can only be used in order to verify the ending state of a transition τ, without causing a controllability problem, if it starts with an input at a port from $P(\tau)$. Thus, it may be necessary to use more than one distinguishing sequence, the different distinguishing sequences starting with input at different ports. Consider, for example, the 2-port FSM M_1 given in Figure 4 that has input alphabet defined by $X_L = \{a, c\}$ and $X_U = \{b\}$ and output alphabet defined by $Y_L = \{2, 3\}$ and $Y_U = \{0, 1\}$. Then $\bar{D}_1 = ba$ and $\bar{D}_2 = ab$ are locally distinguishing sequences M_1, as can be seen from Table 1.

Suppose we wish to use a set $\mathcal{D} = \{\bar{D}_1, \ldots, \bar{D}_r\}$ of distinguishing sequences to check the ending states of the transitions of M. If Υ denotes the transitions of M then $\mathcal{D}$ must satisfy the following.

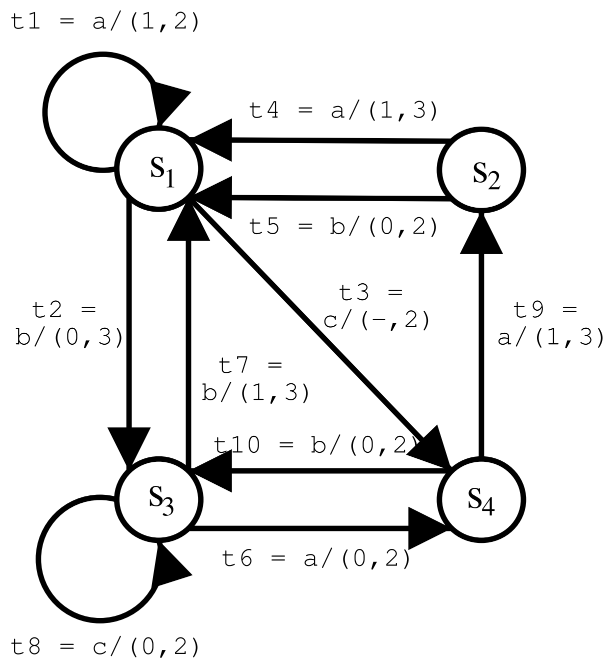

Fig. 4 The 2-port FSM M_1

State	At U for ba	At L for ba	At U for ab	At L for ab
s_1	$b00$	$3a2$	$1b0$	$a23$
s_2	$b01$	$2a2$	$1b0$	$a33$
s_3	$b11$	$3a2$	$0b0$	$a22$
s_4	$b00$	$2a2$	$1b0$	$a32$

Table 1 The responses to ab and ba

Definition 6 The set $\mathcal{D}$ is *complete for M* if for every transition $\tau \in \Upsilon$ there exists some $\bar{D} \in \mathcal{D}$ such that $inport(\bar{D}) \in P(\tau)$.

Given a set K and a set A of subsets of K ($A \subseteq \mathcal{P}(K)$), a set $K' \subseteq K$ is a *hitting set* for A if every set in A contains at least one element of K'. Let $in(\mathcal{D})$ denote $\{inport(\bar{D})|\bar{D} \in \mathcal{D}\}$. Further, let $in(\Upsilon)$ denote $\{P(\tau)|\tau \in \Upsilon\}$. Then the set $\mathcal{D}$ is complete for M if and only if for every $A \in in(\Upsilon)$ there exists some $p \in in(\mathcal{D})$ such that $p \in A$.

Proposition 3 *The set $\mathcal{D}$ is complete for M if and only if $in(\mathcal{D})$ is a hitting set for $in(\Upsilon)$.*

Suppose Z is a minimum size hitting set for $in(\Upsilon)$. Then, any set $\mathcal{D}$ of distinguishing sequences to be used must have size at least $|Z|$. Thus it is desirable to use a set $\mathcal{D}$ of distinguishing sequences with the property that $in(\mathcal{D})$ is a minimum size hitting set for $in(\Upsilon)$. Note that while the problem of finding a minimum size hitting set is NP-complete [30], normally the number of ports will not be large; in such cases it is practical to solve this problem. Throughout this paper we use $\mathcal{D} = \{\bar{D}_1, \ldots, \bar{D}_r\}$ to denote a complete set of distinguishing sequences to be used in checking sequence generation.

4 Overcoming controllability problems

This section shows how, under certain conditions, a synchronizable checking sequence can be produced without the addition of external coordination message exchanges. Under some situations there is no observability problem in which case such a checking sequence is sufficient. An example of such a situation is when there is a global clock and a slow environment. This section is structured in the following way. First, we show how we can generate subsequences that verify that the given distinguishing sequences for M are also distinguishing sequences for the SUT $N = (U, X, Y, \delta_N, \lambda_N, u_1) \in \Phi_M$. We then show

how these subsequences can be used in the construction of a checking sequence by including each transition τ in a context in which we know that its starting state is recognized and τ is followed by a distinguishing sequence.

4.1 Verifying the distinguishing sequences

It might appear that, in order to verify that the elements of $\mathcal{D} = \{\bar{D}_1, \ldots, \bar{D}_r\}$ can be used to identify the states of the SUT $N \in \Phi_M$, it is sufficient to use an input sequence that should lead to the n different responses from N to each distinguishing sequence in $\mathcal{D}$. This would demonstrate that each distinguishing sequence is also a distinguishing sequence in the SUT and so each defines a bijection between the states of M and the states of the SUT.

While such an input sequence is capable of showing that each element of $\mathcal{D}$ is a distinguishing sequence for SUT $N \in \Phi_M$, it need not be able to demonstrate that the elements of $\mathcal{D}$ recognize states of N in a consistent manner; the bijection between states of M and N defined by different distinguishing sequences may differ. For example, there could exist states u and u' of N, states s and s' of M, and $\bar{D}_i, \bar{D}_j \in \mathcal{D}$ such that:

1. N produces $\lambda(s, \bar{D}_i)$ and $\lambda(s', \bar{D}_j)$ in response to $\bar{D}_i$ and $\bar{D}_j$ respectively from state u; and
2. N produces $\lambda(s', \bar{D}_i)$ and $\lambda(s, \bar{D}_j)$ in response to $\bar{D}_i$ and $\bar{D}_j$ respectively from state u'.

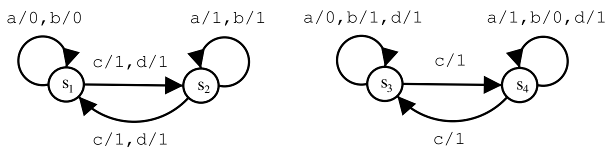

Fig. 5 FSMs M' and M''

Consider, for example, the single-port FSMs M' and M'' shown in Figure 5 and let us suppose that we are testing an SUT that is equivalent to M'' against M'. Here it is clear that a and b are distinguishing sequences of both M' and M''. Let us suppose that we test the SUT with input sequence *acadbcb*. The SUT passes this test because we observe that the output produced by M'' in response to this input sequence is the expected output sequence 0111011. Since the SUT passes this test we must have that a and b are distinguishing sequences for the SUT. However, under the distinguishing sequence a we find that s_3 corresponds to s_1 and s_4 corresponds to s_2 while under the distinguishing sequence b, s_4 corresponds to s_1 and s_3 corresponds to s_2. The two distinguishing sequences thus define different bijections between the states of M' and M'' even though each is a distinguishing sequence for both FSMs.

In constructing a checking sequence on the basis of multiple distinguishing sequences we use the distinguishing sequences to verify the state transition structure of N and thus require that they recognize the states of N in a consistent manner.

Definition 7 Set $\mathcal{D} = \{\bar{D}_1, \ldots, \bar{D}_r\}$ is a *consistent* set of distinguishing sequences for $N \in \Phi_M$ if:

1. Each input sequence in $\mathcal{D}$ is a distinguishing sequence for N; and
2. All r elements of $\mathcal{D}$ recognize the states of N in a consistent manner: i.e. if u is a state of N then there exists a state s of M such that for all $i \in [1, r]$, $\lambda_N(u, \bar{D}_i) = \lambda(s, \bar{D}_i)$.

Definition 8 State u of $N \in \Phi_M$ is *recognized as s by $\mathcal{D}$* if $\mathcal{D}$ is a consistent set of distinguishing sequences for N and u is recognized as state s of M by some $\bar{D}_i \in \mathcal{D}$.

Given the set $\mathcal{D}$ we want to find an input sequence $\bar{x}$ with the property that if the SUT $N \in \Phi_M$ produces the same output sequence as M in response to $\bar{x}$ then we can conclude that $\mathcal{D}$ is a consistent set of distinguishing sequences for N. If this can be done then we can use the elements of $\mathcal{D}$ in the knowledge that if N produces the same output sequence as M in response to $\bar{x}$ then the distinguishing sequences in $\mathcal{D}$ recognize the states of N in a consistent manner.

Definition 9 Input sequence $\bar{x}$ is said to *verify* $\mathcal{D}$ if $\mathcal{D}$ is a consistent set of distinguishing sequences for *every* FSM $N \in \Phi_M$ for which we have that $\lambda_N(u_1, \bar{x}) = \lambda(s_1, \bar{x})$.

The key point in this definition is that since we assume that the SUT N is contained in Φ_M, if $\bar{x}$ verifies $\mathcal{D}$ and we observe the input/output sequence $\bar{x}/\lambda(s_1, \bar{x})$ from the initial state of N then we know that $\mathcal{D}$ must be a consistent set of distinguishing sequences for N. Thus, if we start a test with $\bar{x}$ then there are two possibilities: either we observe a failure or we observe the input/output sequence $\bar{x}/\lambda(s_1, \bar{x})$ and so can conclude that $\mathcal{D}$ is a consistent set of distinguishing sequences for N and so its elements can be used to check the ending states of transitions of N.

Algorithm 1 produces a subsequence that, when included in a path $\bar{\rho}$ from s_1, ensures that the input sequence $\bar{x}$ that labels $\bar{\rho}$ verifies $\{\bar{D}_1\}$ for some $\bar{D}_1 \in \mathcal{D}$. Further subsequences, to verify the remaining elements of $\mathcal{D}$, are then produced in Algorithm 2 using a recursive approach.

Let s^i denote the state $\delta(s_i, \bar{D}_1)$ of M reached from s_i by the input of $\bar{D}_1$. In order to verify $\{\bar{D}_1\}$ we produce a subsequence using the following algorithm.

Algorithm 1 *1. For each state s_i ($1 \leq i < n$) define a transfer sequence $\bar{T}_i^1$ that labels a path of M from s^i to s_{i+1} such that $\bar{D}_1/\lambda(s_i, \bar{D}_1)\bar{T}_i^1$ is the label of a synchronizable path from s_i to s_{i+1} that may be followed by input at port inport$(\bar{D}_1)$ without causing a synchronization problem.*

2. Return the subsequence from s_1 that has label $\bar{D}_1/\lambda(s_1, \bar{D}_1)\bar{T}_1^1\bar{D}_1/\lambda(s_2, \bar{D}_1)\bar{T}_2^1 \ldots \bar{D}_1/\lambda(s_n, \bar{D}_1)$.

The subsequence returned by this process is denoted $\bar{\alpha}_1$. For example, the sequence $\bar{\alpha}_1$ that is obtained by applying Algorithm 1 to 2-port FSM M_1 is formed by the concatenation of the following subsequences: $t_2t_6t_9$; $t_5t_1t_2$; $t_7t_1t_2t_6$; and $t_{10}t_6$.

Recall that Assumption 1 states that for any pair τ, τ' of transitions there is a synchronizable path that starts with τ and ends in τ'.

Proposition 4 *Given an FSM M and a distinguishing sequence $\bar{D}_1$ for M, the subsequence $\bar{\alpha}_1$ can be constructed using $\bar{D}_1$.*

Proposition 5 *The length of the sequence $\bar{\alpha}_1$ produced by Algorithm 1 is of $O(n(n + |\bar{D}_1|))$.*

Proposition 6 *If the label of $\bar{\alpha}_1$ is the label of a path in $N \in \Phi_M$ from some state u of N then $\bar{D}_1$ is a distinguishing sequence for N.*

Proof

This follows since N has at most n states and in $\bar{\alpha}_1$ it produces n different responses to $\bar{D}_1$. □

Having produced a subsequence that verifies $\{\bar{D}_1\}$, we get the following definition of what it means to verify that an element of $\mathcal{D}$ is a distinguishing sequence for N and that it recognizes states of N in the same way as $\bar{D}_1$. Essentially this says that we require that if the SUT passes a test starting with input sequence $\bar{x}$ then every $\bar{D}_i \in \mathcal{D}$ defines the same bijection between the states of the SUT and the states of M.

Definition 10 Sequence $\bar{D}_i \in \mathcal{D}$, $1 < i \leq r$, is *verified relative to* $\bar{D}_1$ by input sequence $\bar{x}$ if the following hold:

1. The response of M to $\bar{x}$ contains n different output sequences produced in response to $\bar{D}_i$; and
2. Whenever an SUT N produces the expected output sequence in response to $\bar{x}$, for every $u \in U$ of N such that $\bar{D}_1/\lambda(s, \bar{D}_1)$ labels a path from u (some state s of M) we have that $\bar{D}_i/\lambda(s, \bar{D}_i)$ labels a path from u.

Proposition 7 *If all the $\bar{D}_i \in \mathcal{D}$ are verified relative to $\bar{D}_1$ by input sequence $\bar{x}$ then $\bar{x}$ verifies $\mathcal{D}$.*

Proof

This simply follows from the fact that if $\bar{D}_i$ is verified relative to $\bar{D}_1$ then $\bar{D}_1$ and $\bar{D}_i$ must define the same bijection between states of M and states of N. □

We now explain how subsequences can be generated to verify the elements of $\mathcal{D} \setminus \{\bar{D}_1\}$ relative to $\bar{D}_1$. We define this process in a recursive manner: we assume that we have produced subsequences that verify $\bar{D}_1, \ldots, \bar{D}_{i-1}$ relative to $\bar{D}_1$ and show how, on the basis of this, $\bar{D}_i$ can be verified relative to $\bar{D}_1$.

The algorithm for producing subsequences that verify some $\bar{D}_i$ relative to $\bar{D}_1$ operates in the following way. For each state s_k we wish to apply $\bar{D}_i$ after a path $\bar{\rho}$ from M whose ending state is s_k and whose starting and ending states have been recognized using a distinguishing sequence already verified. We ensure that the starting state of $\bar{\rho}$ is recognized by beginning it with a distinguishing sequence already verified. We ensure that the ending state of $\bar{\rho}$ has been recognized by including $\bar{\rho}$ followed by $\bar{D}_j/\lambda(s_k, \bar{D}_j)$ for some $\bar{D}_j$ with $j < i$ that has already been verified. We add a further subsequence in the form of $\bar{\rho}$ followed by $\bar{D}_i/\lambda(s_k, \bar{D}_i)$. Since $\bar{D}_j$ has already been verified relative to $\bar{D}_1$, we know that $\bar{D}_i$ is being applied in the state recognized as s_k by $\bar{D}_1$. Note that this procedure requires that we can follow $\bar{\rho}$ with either $\bar{D}_i$ or $\bar{D}_j$ and this places constraints on the ports involved in the final transition of $\bar{\rho}$.

We could apply this procedure for every state s_k of M. However, this is not necessary. Instead, it is sufficient to apply this procedure for $n-1$ states and also apply $\bar{D}_i$ in the remaining state: by observing an nth different response to $\bar{D}_i$ we show that $\bar{D}_i$ is a distinguishing sequence for N and also, by a process of elimination, show that it recognizes the states of N in a manner that is consistent with $\bar{D}_1$.

Algorithm 2 *1. For $i = 2$ to r do*

2. *Choose some subset $S_i \subseteq S$ with size at least $n-1$ such that for every state $s_k \in S_i$ there is some transition $\tau^i_{s_k}$ with ending state s_k and $1 \leq j < i$ such that inport($\bar{D}_i$), inport($\bar{D}_j$) $\in P(\tau^i_{s_k})$. Here $\tau^i_{s_k}$ is any transition that has ending state s_k and can be followed both by the distinguishing sequence $\bar{D}_i$ and a distinguishing sequence $\bar{D}_j$ already considered.*
3. *For all $s_k \in S_i$, choose a state s and produce two synchronizable paths from s with labels $\bar{D}_a/\lambda(s, \bar{D}_a)\bar{T}^i_k \bar{D}_j/\lambda(s_k, \bar{D}_j)$ and $\bar{D}_a/\lambda(s, \bar{D}_a)\bar{T}^i_k\bar{D}_i/\lambda(s_k, \bar{D}_i)$ such that*
   (a) $\bar{T}^i_k$ ends in $\tau^i_{s_k}$; and
   (b) $1 \leq a, j < i$.
   Here $\bar{T}^i_k$ is any input/output sequence that can follow $\bar{D}_a/\lambda(s, \bar{D}_a)$, for some $\bar{D}_a$ already considered, and ends in $\tau^i_{s_k}$.
   Let us assume that these sequences label paths of the SUT N and that the sequences produced in earlier iterations and Algorithm 1 are also paths of N. The key point is that since $\bar{D}_a$ has already been considered, we know that $\bar{D}_a/\lambda(s, \bar{D}_a)\bar{T}^i_k\bar{D}_j/\lambda(s_k, \bar{D}_j)$ and $\bar{D}_a/\lambda(s, \bar{D}_a)\bar{T}^i_k\bar{D}_i/\lambda(s_k, \bar{D}_i)$ must be applied in the same state of N if these are labels of paths in N. In addition, the first of these sequences verifies the node that follows $\bar{D}_a/\lambda(s, \bar{D}_a)\bar{T}^i_k$ and thus for the second we know that $\bar{D}_i$ is being applied in a state of N that is recognized as s_k by $\bar{D}_1$.
4. *If $|S_i| < n$ then let $\{s\} = S \setminus S_i$ and generate a path with label $\bar{D}_i/\lambda(s, \bar{D}_i)$ starting at s.*
5. *end for*

Using Algorithm 2 with $S_2 = \{s_2, s_3, s_4\}$ to 2-port FSM M_1 yields $t_2t_6t_9t_4t_2$ and $t_2t_6t_9t_5t_1$ for s_2, $t_5t_1t_2t_6t_{10}$ and $t_5t_1t_2t_7t_1$ for s_3, $t_7t_1t_2t_6t_9t_5$ and $t_7t_1t_2t_6t_{10}t_6$ for s_4, t_1t_2 for s_1.

We now show that the sequences produced by Algorithms 1 and 2 verify $\mathcal{D}$ and then give a sufficient condition for us to be able to apply these algorithms[6].

Proposition 8 *Suppose that for input sequence $\bar{x}$ we have that $\bar{x}/\lambda(s_1, \bar{x})$ contains the subsequence $\bar{\alpha}_1$ produced by Algorithm 1 and also the set of subsequences produced by Algorithm 2. Then $\bar{x}$ verifies $\mathcal{D}$.*

[6] The proof of Proposition 8 is in the Appendix.

Proposition 9 *Let $d = max\{|\bar{D}_1|, \ldots, |\bar{D}_r|\}$. Then Algorithm 2 returns $O(nr)$ sequences whose total length is of $O(nrd)$.*

The following gives a condition under which Algorithms 1 and 2 can be applied.

Assumption 2 *There is some known ordering $\bar{D}_1, \ldots, \bar{D}_r$ of the elements of $\mathcal{D}$ such that for all $1 < i \leq r$ there is a subset $S_i \subseteq S$ of size at least $n-1$ where for all $s \in S_i$ there exists $1 \leq j < i$ and a transition τ_s^i of M with ending state s such that $inport(\bar{D}_i), inport(\bar{D}_j) \in P(\tau_s^i)$.*

Proposition 10 *If Assumption 2 holds with a given ordering $\bar{D}_1, \ldots, \bar{D}_r$ then Algorithms 1 and 2 produce subsequences that verify the elements of $\mathcal{D}$.*

It is now possible to simplify this condition for the case where there are two ports U and L.

Proposition 11 *Suppose M has ports U and L. Algorithms 1 and 2 produce subsequences that verify the elements of $\mathcal{D}$ if there exist some subset $S' \subseteq S$ of size at least $n-1$ such that for all $s \in S'$ there exists a transition τ of M that ends at s such that $U, L \in P(\tau)$.*

4.2 Producing a checking sequence with no coordination problems

We have seen how a set of subsequences that verify the sequences in $\mathcal{D}$ can be produced. The next problem is to produce subsequences that verify the transitions of SUT $N \in \Phi_M$. These subsequences can be combined, with those produced to verify $\mathcal{D}$, to form a checking sequence.

Let $\tau = (s, s', x/y)$ be a transition of M. Then τ is verified by any synchronizable subsequence from a state u of N recognized as state s that has input portion $x\bar{D}_i$ for some $\bar{D}_i \in \mathcal{D}$. Since $\mathcal{D}$ is complete it is always possible to follow τ by some element of $\mathcal{D}$. Thus, the only remaining issue is how we can apply $x\bar{D}_i$ in a state that is recognized as s. The following shows how this can be achieved.

Algorithm 3 *1. Input a transition $\tau = (s, s', x/y)$ of M.*

2. Produce two synchronizable transition sequences labelled by

(a) The input/output sequence $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}_\tau\bar{D}_c/\lambda(s, \bar{D}_c)$ from state s_i; and

(b) The input/output sequence $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}_\tau x/y\bar{D}_b/\lambda(s', \bar{D}_b)$ from state s_i

where $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}_\tau$ labels a synchronizable path of M from s_i to s and $\bar{D}_a, \bar{D}_b, \bar{D}_c \in \mathcal{D}$.

The first of these sequences checks that $\bar{D}_a/\lambda(s_i, \bar{D}_a)$ $\bar{T}_\tau$ reaches the correct state of the SUT while the second checks that the input of x in this state leads to the expected state of the SUT.

3. Return these two synchronizable transition sequences.

Applying Algorithm 3 to all of the transitions of 2-port FSM M_1 yields the sequences in Table 2.

Transition $\tau = (s, s', x/y)$	For $\bar{D}_a/\lambda(s_i, \bar{D}_a)$ $\bar{T}x/y\bar{D}_b/\lambda(s', \bar{D}_b)$	For $\bar{D}_a/\lambda(s_i, \bar{D}_a)$ $\bar{T}\bar{D}_c/\lambda(s, \bar{D}_c)$
t_1	$t_9t_5t_1t_1t_2$	$t_9t_5t_1t_2$
t_2	$t_9t_5t_2t_6t_{10}$	$t_9t_5t_1t_2$
t_3	$t_9t_5t_3t_9t_5$	$t_9t_5t_1t_2$
t_4	$t_2t_6t_9t_4t_1t_2$	$t_2t_6t_9t_5t_1$
t_5	$t_2t_6t_9t_5t_1t_2$	$t_2t_6t_9t_5t_1$
t_6	$t_6t_{10}t_6t_9t_5$	$t_6t_{10}t_6t_{10}$
t_7	$t_5t_1t_2t_7t_1t_2$	$t_5t_1t_2t_7t_1$
t_8	$t_6t_{10}t_8t_7t_1$	$t_6t_{10}t_6t_{10}$
t_9	$t_{10}t_6t_9t_4t_2$	$t_{10}t_6t_9t_5$
t_{10}	$t_{10}t_6t_{10}t_7t_1$	$t_{10}t_6t_9t_5$

Table 2 Sequences that test the transitions

Proposition 12 *Suppose that $\bar{x}$ is the input portion of the label of a synchronizable path $\bar{\rho}$ in M that verifies $\mathcal{D}$ and contains the subsequences produced by Algorithm 3 when given $\tau = (s, s', x/y)$ as input. If the label of $\bar{\rho}$ is the label of a path in N then N includes a transition from the state recognized as s by $\bar{D}_1$ to the state recognized as s' by $\bar{D}_1$ that has input/output pair x/y.*

Proof

This follows from Proposition 8 and the definition of what it means for $\bar{x}$ to verify $\mathcal{D}$. □

Proposition 13 *Let $d = max\{|\bar{D}_1|, \dots, |\bar{D}_r|\}$. Then the application of Algorithm 3 to all of the transitions of M returns $O(n|X|)$ sequences whose total length is of $O(n|X|d)$.*

Thus, the checking sequence generation algorithm proceeds in the following way.

Algorithm 4 *1. Set $R = \emptyset$.*

2. Generate a path $\bar{\rho}_0$ on the basis of Algorithm 1 and add this to R.

3. Apply Algorithm 2 and add the resultant paths to R.

4. For every transition τ of M, apply Algorithm 3 with τ and add the resultant paths to R.

5. Remove from R every path that is a proper subpath of some other path in R.

6. Choose an (arbitrary) order for the elements of R to get paths $\bar{\rho}_1, \dots, \bar{\rho}_{|R|}$ such that if some element of R starts at state s_1 then $\bar{\rho}_1$ starts at s_1.

7. Produce a synchronizable path $\bar{\rho} = \bar{\rho}'_1\bar{\rho}_1 \dots \bar{\rho}'_{|R|}\bar{\rho}_{|R|}$ of M, where $\bar{\rho}'_1, \dots \bar{\rho}'_{|R|}$ are possibly empty paths of M. If $\bar{\rho}'_1$ is non-empty then it must be chosen so that it starts with a subpath with label $\bar{D}_i/\lambda(s_1, \bar{D}_i)$ for some $\bar{D}_i \in \mathcal{D}$.

8. Return the path $\bar{\rho}$.

The application of Algorithm 4 to 2-port FSM M_1 with the results accumulated earlier yields:

- Step 2: $t_2t_6t_9t_5t_1t_2t_7t_1t_2t_6t_{10}t_6$
- Step 3: $t_2t_6t_9t_4t_2$, $\underline{t_2t_6t_9t_5t_1}$, $t_5t_1t_2t_6t_{10}$, $\underline{t_5t_1t_2t_7t_1}$, $t_7t_1t_2t_6t_9t_5$, $\underline{t_7t_1t_2t_6t_{10}t_6}$, and $\underline{t_1t_2}$. Here we underline a sequence if it is eliminated in Step 5.
- Step 4 : Include the sequences from Table 2.
- Step 5 : Remove from R the paths that are proper subpaths of other paths in R. We remove the paths that are underlined in Table 3 in addition to those indicated above (Step 3).

Transition τ	For $\bar{D}_a/\lambda(s_i,\bar{D}_a)$ $\bar{T}x/y\bar{D}_b/\lambda(s',\bar{D}_b)$	For $\bar{D}_a/\lambda(s_i,\bar{D}_a)$ $\bar{T}\bar{D}_c/\lambda(s,\bar{D}_c)$
t_1	$t_9t_5t_1t_1t_2$	$\underline{t_9t_5t_1t_2}$
t_2	$t_9t_5t_2t_6t_{10}$	$\underline{t_9t_5t_1t_2}$
t_3	$t_9t_5t_3t_9t_5$	$\underline{t_9t_5t_1t_2}$
t_4	$t_2t_6t_9t_4t_1t_2$	$\underline{t_2t_6t_9t_5t_1}$
t_5	$\underline{t_2t_6t_9t_5t_1t_2}$	$\underline{t_2t_6t_9t_5t_1}$
t_6	$t_6t_{10}t_6t_9t_5$	$t_6t_{10}t_6t_{10}$
t_7	$\underline{t_5t_1t_2t_7t_1t_2}$	$\underline{t_5t_1t_2t_7t_1}$
t_8	$t_6t_{10}t_8t_7t_1$	$\underline{t_6t_{10}t_6t_{10}}$
t_9	$t_{10}t_6t_9t_4t_2$	$\underline{t_{10}t_6t_9t_5}$
t_{10}	$t_{10}t_6t_{10}t_7t_1$	$\underline{t_{10}t_6t_9t_5}$

Table 3 The additional sequences

Steps 6, 7, and 8 : The sequences identified are then combined to give a path $\bar{\rho}$ given below where the $\bar{\rho}'_i$, that are added to connect the subsequences, are shown in bold. For readability, the path $\bar{\rho}$ is in several parts; the entire path is formed by concatenating the subsequences in the given order.

$t_2t_6t_9t_5t_1t_2t_7t_1t_2t_6t_{10}t_6$; $t_9t_5t_3t_9t_5$; $t_2t_6t_9t_4t_2$; $t_7t_1t_2t_6t_9t_5$; $t_2t_6t_9t_4t_1t_2$ $\mathbf{t_6}$; $t_{10}t_6t_9t_4t_2$; $t_6t_{10}t_6t_9t_5$ $\mathbf{t_3}$; $t_9t_5t_1t_1t_2$; $t_6t_{10}t_6t_{10}$; $t_6t_{10}t_8t_7t_1$ $\mathbf{t_3}$; $t_9t_5t_2t_6t_{10}$ $\mathbf{t_6t_9}$; $t_5t_1t_2t_6t_{10}$; $\mathbf{t_6}$; $t_{10}t_6t_{10}t_7t_1$.

The following shows that if there are no observability problems then the input portion of the label of the path $\bar{\rho}$ returned by Algorithm 4 is a checking sequence.

Theorem 1 *Let us suppose that $\bar{x}/\bar{y}$ is the label of the path $\bar{\rho}$ returned by Algorithm 4. If $\bar{x}/\bar{y}$ is the label of a path from the initial state of SUT $N \in \Phi_M$ then N is globally equivalent to M.*

Proof

This follows from Proposition 12. □

We can now state the complexity of the test generation process[7].

Theorem 2 *Let us suppose that $\bar{x}/\bar{y}$ is the label of the path $\bar{\rho}$ returned by Algorithm 4. Let $d = max\{|\bar{D}_1|,\ldots,|\bar{D}_r|\}$. Then $\bar{\rho}$ has length of $O(n(n+d)(|X|+r)))$.*

Observe that if we fix the number of ports, and thus fix an upper bound on r, this gives the same complexity as algorithms for producing a checking sequence from a single-port FSM using a distinguishing sequence (see, for example, [19,15,44,23]).

5 Overcoming observability problems

We have seen how, under certain conditions, it is possible to produce a checking sequence that has no controllability problems. This section describes an approach to augmenting this checking sequence for the case where there can be observability problems. First note that, since there can be observability problems, in order to distinguish states it is necessary to locally distinguish them and so we assume that the set $\mathcal{D}$ contains locally distinguishing sequences. Since the problem of checking the output of the transitions without encountering observability problems has already been considered [5–7] we concentrate on the problem of ensuring that the input sequence checks the state transition structure of the SUT.

[7] The proof of Theorem 2 is contained in the Appendix.

Suppose that an input sequence $\bar{D}$ locally distinguishes states s and s' at port p and that, if $\bar{D}$ is input when M is in state s then the sequence $\bar{z}$ is observed at p and if $\bar{D}$ is input when M is in state s' then the sequence $o\bar{z}$ is observed at p for some $o \in Y_p$. Suppose further that, in testing, we follow a transition $\tau = (s_i, s, x/y)$ with input $\bar{D}$ and that $y|_p = o$. Then, if the input of x in state s_i instead leads to output y' that differs from y only at p, where it produces $-$, and moves to s' then the expected sequence $o\bar{z}$ is seen at p. Thus $\bar{D}$ has failed to detect the state transfer fault: this has been masked by an output fault. Naturally, similar problems can occur due to incorrect output after the application of $\bar{D}$.

If we consider the label $\bar{z}$ of a synchronizable path $\bar{\rho}$ of M and the projection $\pi_p(\bar{z})$ observed at port p, this can be represented as $\pi_p(\bar{z}) = \bar{o}_1 x_1 \bar{o}_2 \ldots \bar{o}_k x_k \bar{o}_{k+1}$ for some $\bar{o}_1, \ldots, \bar{o}_{k+1} \in Y_p^*$ and $x_1, \ldots, x_k \in X_p$. Each transition in the path $\bar{\rho}$ has (possibly null) output at p that falls into one of the $\bar{o}_i$ and so for each transition in $\bar{\rho}$ there is a corresponding $\bar{o}_i$. The output sequences $\bar{o}_1, \ldots \bar{o}_{k+1}$ are separated by the inputs $x_1, \ldots, x_k$ at p and so there cannot be undetectable output-shifting faults at p between two transitions whose corresponding subsequences $\bar{o}_i$ and $\bar{o}_j$ are different ($i \neq j$). Naturally, there might be undetectable output-shifting faults between two transitions with the same corresponding subsequence $\bar{o}_i$. This observation inspires the following definition.

Definition 11 Locally distinguishing sequence $\bar{D} = x_1 \ldots x_k$ is *resilient* if for every pair $s, s' \in S$, with $s \neq s'$, there exists a port p and $1 \leq i < j \leq k$, $\bar{D} = \bar{x}'_1 x_i \bar{x}'_2 x_j \bar{x}'_3$ with $x_i, x_j \in X_p$ and $\pi_p(\gamma(\delta(s, \bar{x}'_1), x_i \bar{x}'_2)) \neq \pi_p(\gamma(\delta(s', \bar{x}'_1), x_i \bar{x}'_2))$.

This says that for any pair of states, there must be a port p such that the response to $\bar{D}_i$ differs at p between two inputs at p and thus this difference cannot be masked by previous or following input at p[8].

An important property of a resilient distinguishing sequence $\bar{D}$ is that for an input sequence that should trigger the n responses to $\bar{D}$ allowed by M we have that if an SUT $N \in \Phi_M$ passes this test we must have that not only is $\bar{D}$ a distinguishing sequence for N but it must be a resilient distinguishing sequence for N.

Proposition 14 *Suppose that $\bar{\rho}$ is a path of M starting at s_1 that has input portion $\bar{x}$, $\bar{D}$ is a resilient locally distinguishing sequence for M and $\bar{\rho}$ contains subpaths corresponding to the application of $\bar{D}$ in each of the n states of M. If $\bar{x}$ does not locally distinguish M and $N \in \Phi_M$ then $\bar{D}$ is a resilient locally distinguishing sequence for N.*

Proposition 15 *Let us suppose that $\mathcal{D}$ is a complete set of resilient locally distinguishing sequences and $\bar{x}$ is an input sequence returned by Algorithm 4. If $\pi_p(\gamma(s_1, \bar{x})) = \pi_p(\gamma_N(u_1, \bar{x}))$ for all $p \in [1, m]$ then $\mathcal{D}$ is a consistent set of resilient locally distinguishing sequences for N.*

Proof

[8] The proof of Proposition 14 is in the Appendix.

State	At U for $abab$	At L for $abab$	At U for $baba$	At L for $baba$
s_1	$1b00b0$	$a23a22$	$b00b00$	$3a22a2$
s_2	$1b00b0$	$a33a22$	$b01b00$	$2a23a2$
s_3	$0b00b0$	$a22a22$	$b11b00$	$3a23a2$
s_4	$1b01b0$	$a32a23$	$b00b00$	$2a22a2$

Table 4 The responses to *abab* and *baba*

First note that by Proposition 14, if $N \in \Phi_M$ and M are not locally distinguished by $\bar{x}$ then each element of $\mathcal{D}$ is a resilient locally distinguishing sequence for N. The result thus follows in a similar manner to Proposition 8. □

Assumption 3 *The elements of $\mathcal{D}$ are resilient locally distinguishing sequences.*

It is clear that a locally distinguishing sequence need not be resilient. The set $\mathcal{D}$ given earlier for the 2-port FSM M_1 does not satisfy Assumption 3. However, as we can see in Table 4, the sequences *abab* and *baba* do satisfy Assumption 3.

Suppose that Algorithm 4 is applied using a set $\mathcal{D}$ of resilient locally distinguishing sequences and returns path $\bar{\rho}$ whose label has input portion $\bar{x}$. We define a property of the SUT $N \in \Phi_M$ and prove that this must hold if $\bar{x}$ does not locally distinguish N and M.

Definition 12 SUT $N \in \Phi_M$ has *the same transition structure* as M if there is a bijection f from the states of N to the states of M such that:

1. $f(u_1) = s_1$.
2. If u is a state of N and there is a transition from u to u' in N with input x then M has a transition from $f(u)$ to $f(u')$ with input x.

The input sequence produced by Algorithm 4 checks the transition structure of N[9].

Theorem 3 *Suppose that Algorithm 4 is applied using a set $\mathcal{D}$ of resilient locally distinguishing sequences and returns path $\bar{\rho}$ whose label has input portion $\bar{x}$. If $\bar{x}$ does not locally distinguish SUT $N \in \Phi_M$ from M then N has the same transition structure as M.*

It is now sufficient to add sequences that check the output produced by each transition τ at each port p. The following definition captures this requirement.

Definition 13 An input sequence $\bar{x}$ *checks the outputs* of M if $N \in \Phi_M$ is globally equivalent to M whenever the following hold

1. N has the same transition structure as M; and
2. $\pi_p(\gamma(s_1, \bar{x})) = \pi_p(\gamma_N(u_1, \bar{x}))$ for all $p \in [1, m]$.

The following shows that even if there are observability and controllability problems then we can augment the sequences produced in Algorithm 4 with sequences that check the output of M to form a checking sequence.

Theorem 4 *If $\bar{x}$ is an input sequence that checks the output of M and starts with the label of a path of M produced by Algorithm 4 using resilient locally distinguishing sequences then $\bar{x}$ is a checking sequence that has no controllability or observability problems.*

[9] The proof of Theorem 3 is contained in the Appendix

Proof

This result follows from Theorem 3 and Definition 13. □

6 Conclusions and Discussion

In the distributed test architecture a tester is placed at each port of the SUT N. If the individual testers cannot communicate with each other then the presence of multiple testers introduces additional controllability and observability problems. It is then important that any checking sequence that we intend to use is free from such problems.

This paper is the first to show how a single checking sequence can be produced for a multi-port FSM without the use of either a reliable reset operation or external coordination messages. Since, in general, such a checking sequence need not exist we introduce conditions to be placed on the specification M under which our algorithm returns checking sequences. If the distributed test architecture is to be used then these could be seen as testability conditions that might be designed into a system.

This paper focused on the generation of checking sequences since such sequences are guaranteed to provide full fault coverage under the assumption that the SUT contains no extra states. Algorithms for generating a checking sequence for a single-port FSM use distinguishing sequences, unique input/output sequences, or a characterization set to verify states of the SUT. In this paper we used distinguishing sequences since even for single-port FSMs there is no checking sequence generation algorithm that uses the alternative approaches and returns a checking sequence of length that is polynomial in terms of the number of states.

First, we investigated the situation in which there are no observability problems. This is the case, for example, when there is a global clock and the SUT responds to inputs sufficiently quickly so that the next input is not applied until after all of the outputs from the previous inputs have been observed. In such a case observability problems can be overcome by the testers timestamping the events they see and so there are no additional observability problems. We showed how multiple distinguishing sequences can be used in forming a checking sequence that does not suffer from controllability problems.

If there are observability problems then these can lead to fault masking and thus to incorrect output not being observed. We showed how the checking sequence can be extended to create a checking sequence that does not suffer from either controllability or observability problems.

This paper has shown how checking sequences can be produced for multi-port FSMs. There remain four main avenues for future work. There is the question as to whether the conditions given in this paper, under which checking sequences are produced, can be weakened. Another question is how to optimize the resultant checking sequence such that significantly shorter checking sequences can be constructed. This may be achieved by solving an optimization problem posed considering the following. First, the selection of the transition sequences used to verify the distinguishing sequences. The second issue is the selection of the subsets and the choice of distinguishing sequences used in forming paths to verify the distinguishing sequences. Similar choices are needed in the generation of transition sequences to verify the transitions and additional choices are required when considering potential observability problems. There is also the issue of how we can produce a minimal length sequence that contains the necessary subsequences. There is the issue of generating resilient locally distinguishing sequences for which a possibly breadth-first search can be used. There may also be scope in adding input to the end of a locally distinguishing sequence in order to make it resilient. Finally, distributed systems are often nondeterministic and it would thus be interesting to extend the approach to such systems, potentially by either using methods such as deterministic testing (see, for example, [18,26,34,35]) in order to ensure that the SUT is deterministic in testing or by using methods from the area of testing from nondeterministic FSMs (see, for example [21,27,38–40]).

Acknowledgements We would like to thank Dr Jessica Chen for many useful discussion around this topic. This work was supported in part by Leverhulme Trust grant number F/00275/D, Natural Sciences and Engineering Research Council (NSERC) of Canada grant number OGP00000976, Testing State Based Systems, and Engineering and Physical Sciences Research Council grant number GR/R43150, Formal Methods and Testing (FORTEST).

References

1. M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Towards a tool environment for model-based testing with AsmL. In *Formal Approaches to Testing*, volume 2931 of *Lecture Notes in Computer Science*, pages 252–266, Montreal, Canada, 2003. Springer-Verlag.
2. S. Boyd and H. Ural. The synchronization problem in protocol testing and its complexity. *Information Processing Letters*, 40(3):131–136, 1991.
3. B. Broekman and E. Notenboom. *Testing Embedded Software.* Addison–Wesley, London, 2003.
4. L. Cacciari and O. Rafiq. Controllability and observability in distributed testing. *Information and Software Technology*, 41(11–12):767–780, 1999.
5. J. Chen, R. M. Hierons, and H. Ural. Conditions for resolving observability problems in distributed testing. In *24rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, volume 3235 of *Lecture Notes in Computer Science*, pages 229–242. Springer–Verlag, 2004.
6. J. Chen, R. M. Hierons, and H. Ural. Resolving observability problems in distributed test architectures. In *25th IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, volume 3731 of *Lecture Notes in Computer Science*, pages 219–232. Springer–Verlag, 2005.
7. J. Chen, R. M. Hierons, and H. Ural. Overcoming observability problems in distributed test architectures. *Information Processing Letters*, 98(5):177–182, 2006.

8. W. Chen and H. Ural. Synchronizable checking sequences based on multiple UIO sequences. *IEEE/ACM Transactions on Networking*, 3:152–157, 1995.

9. Karnig Derderian, Robert M. Hierons, Mark Harman, and Qiang Guo. Input sequence generation for testing of communicating finite state machines (cfsms). In *Genetic and Evolutionary Computation (GECCO 2004)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1429–1430, Seattle, WA, USA, 2004. Springer.

10. Rita Dorofeeva, Nina Yevtushenko, Khaled El-Fakih, and Ana R. Cavalli. Experimental evaluation of fsm-based testing methods. In *Third IEEE International Conference on Software Engineering and Formal Methods (SEFM 2005)*, pages 23–32. IEEE Computer Society, 2005.

11. R. Dssouli and G. von Bochmann. Error detection with multiple observers. In *Protocol Specification, Testing and Verification V*, pages 483–494. Elsevier Science (North Holland), 1985.

12. R. Dssouli and G. von Bochmann. Conformance testing with multiple observers. In *Protocol Specification, Testing and Verification VI*, pages 217–229. Elsevier Science (North Holland), 1986.

13. E. Farchi, A. Hartman, and S. Pinter. Using a model-based test generator to test for standard conformance. *IBM systems journal*, 41(1):89–110, 2002.

14. A. Gill. *Introduction to The Theory of Finite State Machines*. McGraw–Hill, New York, 1962.

15. G. Gonenc. A method for the design of fault detection experiments. *IEEE Transactions on Computers*, 19:551–558, 1970.

16. W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. In *Proceedings of the ACM SIGSOFT Symposium on Software Testing and Analysis*, pages 112–122, 2002.

17. S. Guyot and H. Ural. Synchronizable checking sequences based on UIO sequences. In *Protocol Test Systems, VIII*, pages 385–397, Evry, France, September 1995. Chapman and Hall.

18. Craig Harvey and Paul A. Strooper. Testing java monitors through deterministic execution. In *13th Australian Software Engineering Conference (ASWEC 2001)*, pages 61–67. IEEE Computer Society, 2001.

19. F. C. Hennie. Fault–detecting experiments for sequential circuits. In *Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, Princeton, New Jersey, November 1964.

20. R. M. Hierons. Minimizing the number of resets when testing from a finite state machine. *Information Processing Letters*, 90(6):287–292, 2004.

21. R. M. Hierons. Testing from a non-deterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, 53(10):1330–1342, 2004.

22. R. M. Hierons and H. Ural. Redefining testing (from a finite state machine) in the distributed test architecture. *submitted*.

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

24. R. M. Hierons and H. Ural. UIO sequence based checking sequences for distributed test architectures. *Information and Software Technology*, 45(12):793–803, 2003.

25. J. E. Hopcroft. An n log n algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The theory of Machines and Computation*, pages 189–196. Academic Press, 1971.

26. Gwan-Hwan Hwang, Kuo-Chung Tai, and Ting-Lu Huang. Reachability testing: an approach to testing concurrent software. In *First Asia-Pacific Software Engineering Conference*, pages 246–255, 1994.

27. I. Hwang, T. Kim, S. Hong, and J. Lee. Test selection for a nondeterministic FSM. *Computer Communications*, 24(12):1213–1223, 2001.

28. 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.

29. ITU-T. *Recommendation Z.500 Framework on formal methods in conformance testing.* International Telecommunications Union, Geneva, Switzerland, 1997.

30. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations.* Plenum Press, New York–London, 1972. 85–103.

31. A. Khoumsi. A temporal approach for testing distributed systems. *IEEE Transactions on Software Engineering*, 28(11):1085–1103, 2002.

32. D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.

33. D. Lee and M. Yannakakis. Principles and methods of testing finite–state machines – a survey. *Proceedings of the IEEE*, 84(8):1089–1123, 1996.

34. Yu Lei and Richard H. Carver. A new algorithm for reachability testing of concurrent programs. In *16th International Symposium on Software Reliability Engineering (ISSRE 2005)*, pages 346–355. IEEE Computer Society, 8–11 November 2005.

35. Yu Lei and Richard H. Carver. Reachability testing of concurrent programs. *IEEE Transactions on Software Engineering*, 32(6):382–403, 2006.

36. G. Luo, R. Dssouli, and G. v. Bochmann. Generating synchronizable test sequences based on finite state machine with distributed ports. In *The 6th IFIP Workshop on Protocol Test Systems*, pages 139–153. Elsevier (North-Holland), 1993.

37. G. Luo, R. Dssouli, G. v. Bochmann, P. Venkataram, and A. Ghedamsi. Test generation with respect to distributed interfaces. *Computer Standards and Interfaces*, 16:119–132, 1994.

38. G. Luo, A. Petrenko, and G. v. Bochmann. Selecting test sequences for partially-specified nondeterministic finite state machines. In *The 7th IFIP Workshop on Protocol Test Systems*, pages 95–110, Tokyo, Japan, November 8–10 1994. Chapman and Hall.

39. G. L. Luo, G. v. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–161, 1994.

40. A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das. Nondeterministic state machines in protocol conformance testing. In *Proceedings of Protocol Test Systems, VI (C–19)*, pages 363–378, Pau, France, 28-30 September 1994. Elsevier Science (North-Holland).

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

42. B. Sarikaya and G. v. Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, 32:389–395, April 1984.

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

44. H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1):93–99, 1997.

45. W.-J. Wu, W.-H. Chen, and C. Y. Tang. Synchronizable test sequence for multi–party protocol conformance

testing. *Computer Communications*, 21(13):1177–1183, 1998.

46. M. Yao, A. Petrenko, and G. v. Bochmann. Conformance testing of protocol machines without reset. In *Protocol Specification, Testing and Verification, XIII (C–16)*, pages 241–256. Elsevier (North–Holland), 1993.
47. Y. C. Young and K. C. Tai. Observational inaccuracy in conformance testing with multiple testers. In *IEEE 1st workshop on application-specific software engineering and technology*, pages 80–85, 1998.

Appendix

Proof of Proposition 8

By Proposition 7 it is sufficient to prove that for all $i \in [1, r]$, $\bar{x}$ verifies $\bar{D}_i$ relative to $\bar{D}_1$. Proof by induction on i: the base case with $i = 1$ follows immediately. Inductive hypothesis: for $i < h$ $(h \leq r)$, $\bar{x}$ verifies $\bar{D}_i$ relative to $\bar{D}_1$. It is now sufficient to prove that $\bar{x}$ verifies $\bar{D}_h$ relative to $\bar{D}_1$.

First observe that the subsequences include the input/output sequence $\bar{D}_h/\lambda(s, \bar{D}_h)$ for each state s of M and thus, if contained in the label of a path of N, verify that $\bar{D}_h$ is a distinguishing sequence for N. Now consider some state u of N that is recognized as state s_k of M by $\bar{D}_1$. There are two cases to consider.

1. $s_k \in S_h$. Then we have two paths with labels $\bar{D}_a/\lambda(s, \bar{D}_a)\bar{T}_k^h\bar{D}_j/\lambda(s_k, \bar{D}_j)$ and $\bar{D}_a/\lambda(s, \bar{D}_a)\bar{T}_k^h\bar{D}_h/\lambda(s_k, \bar{D}_h)$ for some $s \in S$ and $\bar{D}_a$ and $\bar{D}_j$ with $1 \leq a, j < h$. By the inductive hypothesis, if these input/output subsequences label paths of N then their input portions are applied in a state u of N recognized as s by $\bar{D}_1$ and the path from u whose label is the input/output sequence $\bar{D}_a/\lambda(s, \bar{D}_a)\bar{T}_k^h$ takes N to a state u_k recognized as s_k by $\bar{D}_j$ and thus by $\bar{D}_1$. Thus, $\bar{D}_h/\lambda(s_k, \bar{D}_h)$ labels a path from u_k as required.
2. $s_k \notin S_h$. First note that every other state u of N is recognized as some unique state s of M by both $\bar{D}_h$ and $\bar{D}_1$. The result follows from observing that there is one remaining output sequence $\lambda(s_k, \bar{D}_h)$ produced by N in response to $\bar{D}_h$; by a process of elimination this can only occur at a state u_k recognized as s_k by $\bar{D}_1$.

$\square$

Proof of Theorem 2

By Proposition 9, Algorithm 2 returns $O(nr)$ sequences with total length of $O(nrd)$. By Proposition 13, Algorithm 3 returns $O(n|X|)$ paths whose total length is of $O(n|X|d)$. Thus $\bar{\rho}$ is produced by connecting $O(nr + n|X|)$ paths whose total length is of $O(nrd + n|X|d)$. Each path added to connect the paths in R has length of $O(n)$ and thus the total length of the paths added to connect those in R is of $O(n(nr+n|X|))$. Thus $\bar{\rho}$ has length of $O(n(nr+n|X|)+nrd+n|X|d) = O((n+d)(nr+n|X|)$.

$\square$

Proof of Proposition 14

Let $\bar{x}_1, \ldots, \bar{x}_n$ denote prefixes of $\bar{x}$ such that in $\bar{\rho}$ each is followed by the input of $\bar{D}$ and $\delta(s_1, \bar{x}_\alpha) = s_\alpha$ $(1 \leq \alpha \leq n)$. Let u_α denote the state $\delta_N(u_1, \bar{x}_\alpha)$ of N

$(1 \leq \alpha \leq n)$. Consider arbitrary states u_α and u_β with $1 \leq \alpha < \beta \leq n$.

Since $\bar{D} = x_1 \ldots x_k$ is a resilient locally distinguishing sequence for M there exists port p and $1 \leq i < j \leq k$, $\bar{D} = \bar{x}'_1 x_i \bar{x}'_2 x_j \bar{x}'_3$ with $x_i, x_j \in X_p$ and $\pi_p(\gamma(\delta(s_\alpha, \bar{x}'_1), x_i\bar{x}'_2))$ $\neq \pi_p(\gamma(\delta(s_\beta, \bar{x}'_1), x_i\bar{x}'_2))$. Since M and N are not locally distinguished by $\bar{x}$, the response of N to $\bar{D}$ in states u_α and u_β must include the substrings $\pi_p(\gamma(\delta(s_\alpha, \bar{x}'_1), x_i\bar{x}'_2))$ and $\pi_p(\gamma(\delta(s_\beta, \bar{x}'_1), x_i\bar{x}'_2))$ respectively after the prefix $\bar{x}'_1$ of $\bar{D}$. Further, these subsequences start with and are followed by input at p. Thus, u_α and u_β are locally distinguished by $\bar{D}$ as required. By the definition of a locally distinguishing sequence being resilient, since this holds for every pair of distinct states of N, $\bar{D}$ is a resilient locally distinguishing sequence for N. □

Proof of Theorem 3

By Proposition 15, $\mathcal{D}$ is a consistent set of resilient locally distinguishing sequences for N. Define a function f from S to U by: given $u \in U$ of N, $f(u) = s$ if and only if u is reached by some prefix $\bar{x}_u$ of $\bar{x}$ such that $\delta(s_1, \bar{x}_u) = s$ and $\bar{x}_u$ is followed by an element of $\mathcal{D}$ in $\bar{x}$. Since N has no more states than M and $\mathcal{D}$ is a consistent set of resilient locally distinguishing sequences for both M and N, f is a bijection. Proof by contradiction: suppose that $\bar{x}$ does not locally distinguish N from M and N does not have the same transition structure as M.

Since $\bar{x}$ does not locally distinguish N from M and $\bar{x}$ starts with some $\bar{D}_i \in \mathcal{D}$, $f(u_1) = s_1$. Thus, since N does not have the same transition structure as M, there is a state u of N and a transition from u to u' in N with input x such that M does not have a transition from $f(u)$ to $f(u')$ with input x. Let $s = f(u)$, $s' = f(u')$, and $s'' = \delta(s, x)$ $(s' \neq s'')$.

Applying $\bar{x}$ to M produces a synchronizable transition sequence that includes subsequences defined by:

1. The input/output sequence $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}x/y\bar{D}_b/\lambda(s'', \bar{D}_b)$ from state s_i; and
2. The input/output sequence $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}\bar{D}_c/\lambda(s, \bar{D}_c)$ from state s_i

for some output y and state s_i of M such that $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}$ labels a path from s_i to s. Since the distinguishing sequences in $\mathcal{D}$ are resilient and $\bar{x}$ does not locally distinguish M and N, in N the sequence $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}$ labels a path from the state u_i of N with $f(u_i) = s_i$ to u. Further, since $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}x/y\bar{D}_b/\lambda(s'', \bar{D}_b)$ labels a path in N, $\bar{D}_a/\lambda(s_i, \bar{D}_a)\bar{T}x/y$ goes from u_i to the state u'' of N with $f(u'') = s''$. Thus, if N receives input x when in state u it moves to state u''. Since N is deterministic, $u' = u''$ and so $s' = s''$. This provides a contradiction as required. □