Controllable testing from Nondeterministic Finite State Machines with Multiple Ports

Robert M. Hierons *Senior Member, IEEE*

***Abstract*— Some systems have physically distributed interfaces, called ports, at which they interact with their environment. We place a tester at each port and if the testers cannot directly communicate and there is no global clock then we are using the distributed test architecture. It is known that this test architecture introduces controllability problems when testing from a deterministic finite state machine. This paper investigates the problem of testing from a nondeterministic finite state machine in the distributed test architecture and explores controllability. It shows how we can decide in polynomial time whether an input sequence is controllable. It also gives an algorithm for generating such an input sequence $\bar{x}$ and shows how we can produce testers that implement $\bar{x}$.**

***Index Terms*— D2.4: Software Engineering/Software/Program Verification, D2.5: Software Engineering/Testing and Debugging, finite state machine, nondeterminism, controllability, distributed test architecture.**

I. Introduction

Reactive systems are state-based and are often modelled using finite state machines (FSMs) or languages such as statecharts and SDL based on extended finite state machines (EFSMs). Since FSM based test techniques can be applied when testing from EFSMs there has been much interest in testing from FSMs ([1], [2], [3]). Some reactive systems have physically distributed interfaces, called ports, at which they interact with their environment. Such systems can be modelled as multi-port FSMs with each port having input and output alphabets/sets. In testing we place a tester at each port. If the testers cannot directly communicate with one another during testing and there is no global clock then we are testing in the distributed test architecture and a tester observes only the interactions at its port.

We can have controllability problem when testing from a deterministic FSM (DFSM) in the distributed test architecture [4], [5], [6], [7], [8], [9]. Consider, for example, an input sequence that starts with x_1 at port p that should lead to output y_p at p and which we wish to follow with x_2 at port $q \neq p$. The tester at q does not observe either x_1 or y_q and so does not know when to apply x_2: there is a *controllability problem*. There can also be fault masking (*observability problems*) since each tester only observes events at its port and in general it is not possible to reconstruct the global sequence that occurred. Previous work on testing from an FSM in the distributed test architecture has considered DFSMs. This work has investigated methods for producing input sequences that have no controllability problems and for overcoming possible fault masking. Only recently have new conformance relations been defined to recognise the reduced observational power of testing [10].

Given that nondeterminism aids abstraction and can arise through a system being distributed, the restriction to DFSMs is a significant limitation. This paper considers the problem of testing from a nondeterministic FSM. The underlying notion of only observing projections of observations has been investigated in the context of refinement of CSP [11], although the technical issues are different. There has also been work on avoiding a different type of controllability problem that causes races when a component is embedded in a context [12].

This paper makes the following contributions. Section III explores testing from an FSM in the distributed test architecture. In Section IV we define what it means for an input sequence to have a controllability problem. In Section V we explain how one can decide whether an input sequence has controllability problems and show how this can be used to drive test generation. We also explain how we can take an input sequence, with no controllability problems, and produce testers that implement this. The concept of a test being controllable has recently been explored in the context of testing

R.M. Hierons is with the School of Information Systems, and Computing Mathematics, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK

from an input output transition system (IOTS) [13]. However, this work considered a restricted type of IOTS (a transition cannot send output to more than one port) and the algorithms in [13] for deciding whether an input sequence $\bar{x}$ is controllable and producing testers require the construction of all possible responses of M to $\bar{x}$ and this set may contain exponentially many sequences. In contrast, the algorithms given in this paper operate in time that is polynomial in terms of the number of transitions of M and the length of $\bar{x}$.

II. Preliminaries

A. Test sequences

We use X for the set of inputs of the *system under test (SUT)* and Y for the set of outputs. The application of an input sequence from X^* leads to an *input/output sequence* called a *trace*. We let ϵ denote the empty sequence. Trace $x_1/y_1 \ldots x_k/y_k$ (for $1 \leq i \leq k, x_i \in X, y_i \in Y$) can be represented by $\bar{x}/\bar{y}$ where $\bar{x} = x_1 \ldots x_k$ is the *input portion* of $\bar{x}/\bar{y}$ and $\bar{y} = y_1 \ldots y_k$. Trace $\bar{z}$ is a prefix of $x_1/y_1, \ldots, x_k/y_k$ if either $\bar{z} = \epsilon$ or $\bar{z} = x_1/y_1, \ldots, x_i/y_i$ for some $1 \leq i \leq k$. Given set A of sequences, $pre(A)$ denotes the set of prefixes of sequences in A.

B. Multi-port Finite State Machines

A multi-port system has distributed interfaces called ports and we place a tester at each port. We assume that there are $m > 1$ ports with set $\mathcal{P} = \{1, \ldots, m\}$ of names. The distributed test architecture was introduced in protocol conformance testing with two testers: an upper tester and a lower tester (see, for example, [14]). Thus, in the examples we use two ports that we call U and L.

Definition 1 *A (possibly nondeterministic) multi-port finite state machine M with m ports is defined by a tuple (S, s_0, X, Y, T) in which*

1) *S is a finite set of states;*
2) *$s_0 \in S$ is the initial state;*
3) *$X = X_1 \cup \ldots \cup X_m$ is the finite input alphabet in which for all $p \in \mathcal{P}$, X_p is the set of inputs that can be received at p. For all $1 \leq p < q \leq m$, $X_p \cap X_q = \emptyset$: if the SUT can receive the same inputs at different ports we add labels in order to ensure that the X_p are disjoint;*
4) *$Y = (Y_1 \cup \{-\}) \times \ldots \times (Y_m \cup \{-\})$ is the finite output alphabet, where for all $p \in \mathcal{P}$, Y_p denotes the set of outputs the SUT can send to port p and $-$ denotes no output being sent to p. $(y^1, \ldots, y^m) \in Y$ denotes output y^p being sent to port p ($1 \leq p \leq m$); and*
5) *T is the set of transitions, each transition being of the form $(s, s', x/y)$ for $s, s' \in S$, $x \in X$, and $y \in Y$.*

Throughout this paper we use the following notation, sometimes with subscripts: An input sequence will be denoted $\bar{x}$, a trace $\bar{z}$, an input x, an input at port p x^p, an output y and an output at port p y^p.

Transition $(s, s', x/y) \in T$ means that if M receives x when in state s then it can output y and move to s'. A sequence of consecutive transitions $\bar{\rho} = t_1, \ldots, t_k$, $t_i = (s_i, s_{i+1}, x_i/y_i)$, is a *path* with *label* $x_1/y_1, \ldots, x_k/y_k$ and *starting state* s_1. FSM M defines the regular language $\mathcal{L}(M)$ of labels of paths that have starting state s_0. Given input sequence $\bar{x}$, $M(\bar{x})$ denotes the set of traces from $\mathcal{L}(M)$ with input portion $\bar{x}$. A multi-port finite state machine will be called a *finite state machine (FSM)* and when we wish to refer to an FSM with one port we call it a *single-port FSM*.

Figure 1 shows an FSM called M_1. Here a transition $(s_i, s_j, x/y)$ is represented by an arc from s_i to s_j with label x/y. For example, the arc from s_2 to s_1 with label $x^L/(y^U, -)$ represents transition $(s_2, s_1, x^L/(y^U, -))$; if M_1 receives x^L when in state s_2 then it can move to s_1 and output y^U at U.

Given $y = (y^1, \ldots, y^m) \in Y$ and $p \in \mathcal{P}$ we let $y|_p$ denote y^p. Given input x we let $port(x)$ denote the port at which x is input and for output y we let $ports(y)$ denote the set of $p \in \mathcal{P}$ such that $y|_p \neq -$. For an input/output pair x/y we let $ports(x/y)$ denote the set $\{port(x)\} \cup ports(y)$ of ports that are *involved* in x/y and for transition $t = (s, s', x/y)$ we let $ports(t) = ports(x/y)$.

If for every $s \in S$ and $x \in X$ there is at most one transition with starting state s and a label with input portion x then M is a *deterministic* FSM (*DFSM*); otherwise it is *nondeterministic*. If for every $s \in S$ and $x \in X$ there is at least one transition with starting state s and a label that has input portion x then M is *completely-specified*. In this paper we assume that specifications and implementations are completely-specified FSMs but otherwise do not require FSMs to be completely-specified.

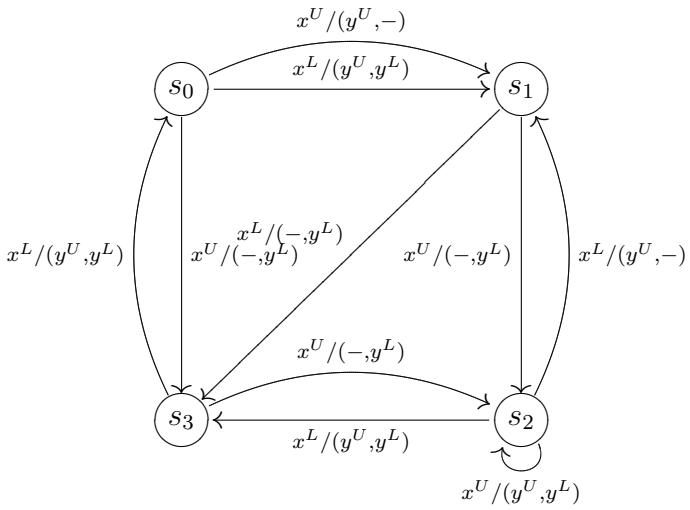

Fig. 1. The FSM M_1

III. Test architectures

In the ISO distributed test architecture [14] there is a tester at each port of the SUT, the testers cannot communicate with one another during testing, and there is no global clock. The distributed test architecture is simple to implement: it does not require the testers to interact during testing. However, we do allow the observations made at the separate ports to be brought together later. It is known that this test architecture can introduce controllability problems when testing from a DFSM [5][1].

In the distributed test architecture the tester at $p \in \mathcal{P}$ only observes the events at p. Given trace $\bar{z}$ and port $p \in \mathcal{P}$ we let $\pi_p(\bar{z})$ denote the projection of $\bar{z}$ at p. This can be defined by the following in which $y^p = y|_p$: $\pi_p(\epsilon) = \epsilon$; if $x \in X_p$ and $y^p \neq -$ then $\pi_p(x/y\bar{z}) = xy^p\pi_p(\bar{z})$; if $x \in X_p$ and $y^p = -$ then $\pi_p(x/y\bar{z}) = x\pi_p(\bar{z})$; if $x \notin X_p$ and $y^p \neq -$ then $\pi_p(x/y\bar{z}) = y^p\pi_p(\bar{z})$; if $x \notin X_p$ and $y^p = -$ then $\pi_p(x/y\bar{z}) = \pi_p(\bar{z})$. For example, $\pi_U(x^U/(y^U, y^L)) = x^U y^U$ and $\pi_L(x^U/(y^U, y^L)) = y^L$. If for all $p \in \mathcal{P}$ we have that $\pi_p(\bar{z}_1) = \pi_p(\bar{z}_2)$ then we write $\bar{z}_1 \sim \bar{z}_2$ and we cannot distinguish between $\bar{z}_1$ and $\bar{z}_2$ in testing. Given a set Z of traces and port p we let $\pi_p(Z)$ denote the set $\{\bar{z} | \exists \bar{z}' \in Z.\bar{z} = \pi_p(\bar{z}')\}$ of projections of traces from Z.

When testing from a single-port FSM M, an FSM N with the same input and output alphabets as M is a *reduction* of M if $\mathcal{L}(N) \subseteq \mathcal{L}(M)$. In the distributed test architecture we need to adapt this notion of conformance and define the notion of local reduction, which is equivalent to but simpler than the relation $dioco$ recently defined for IOTSs [15].

Definition 2 *Given completely-specified FSMs M and N with the same input and output alphabets, N is a* local reduction *of M if for every $\bar{z}_1 \in L(N)$ there exists $\bar{z}_2 \in L(M)$ such that $\bar{z}_1 \sim \bar{z}_2$. Further, N is* locally equivalent *to M if M is a local reduction of N and N is a local reduction of M.*

When testing from a nondeterministic FSM we can have a set Z of allowed traces in response to input sequence $\bar{x}$ and the tester at port p expects to observe an element of $Z_p = \pi_p(Z)$. We could have the tester at p produce verdict pass if and only if it observes an element of Z_p. Let us suppose, however, that $\bar{x} = x_1^U$ and Z contains $x_1^U/(y_1^U, y_1^L)$ and $x_1^U/(y_2^U, y_2^L)$ with $y_1^U \neq y_2^U$ and $y_1^L \neq y_2^L$. If the tester at U observes $x_1^U y_1^U$ and the tester at L observes y_2^L then each returns verdict pass. However, the set of observations is not consistent with any trace in Z; a failure has occurred. Thus, the testers should log their observations and later a failure is declared if there is no trace of the specification with this set of projections.

In this paper we assume that we are testing to determine whether N is a local reduction of M. However, results and definitions regarding controllability do not depend on the conformance relation used.

IV. Controllability problems

When testing from a DFSM a controllability problem occurs when the next input is to be applied at a port p such that the tester at p was not involved in the previous transition. Let us suppose, for example, that the tester at U should apply input x^U but that the previous transition involved input x^L at L and output y^L at L only. The tester at U cannot know when to apply x^U. There has been much interest in controllability problems for DFSMs and here we explore controllability problems when testing from a nondeterministic FSM.

When considering a DFSM we can choose controllable paths. Path $\bar{\rho} = t_1 \dots t_k$, $t_i = (s_i, s_{i+1}, x_i, y_i)$ is *controllable* if for all $1 < i \leq k$, $port(x_i) \in ports(t_{i-1})$. We also say that

[1]If we can connect the testers using a network we can sometimes overcome these problems using external coordination messages but this is not always feasible, especially if there are timing constraints.

$x_1/y_1 \ldots x_k/y_k$ is *controllable*. All sequences of length 0 and 1 are controllable. In M_1, $x^U/(y^U,-)$ $x^U/(-,y^L)x^L/(y^U,y^L)$ is controllable. However, $x^U/(y^U,-)x^L/(-,y^L)$ is not since the second input is at port L but the first input/output pair does not involve L.

When testing from a DFSM we can choose a controllable path $\bar{\rho}$ with starting state s_0 and apply the input portion of its label. However, for an FSM M there may be more than one path that can be triggered by an input sequence. Given input sequence $\bar{x}$ we require that there are no controllability problems in any element of $M(\bar{x})$. However, consider $\bar{x} = x^U x^U x^L$ and M_1: $M_1(\bar{x})$ contains controllable traces $x^U/(y^U,-)x^U/(-,y^L)x^L/(y^U,y^L)$ and $x^U/(-,y^L)$ $x^U/(-,y^L)x^L/(y^U,y^L)$. In the first the tester at L applies x^L after y^L and in the second it applies x^L after $y^L y^L$. If the tester at L observes y^L then it does not know whether to apply x^L or to wait for another y^L: there is a controllability problem. Each tester can only make a decision regarding when to send inputs on the basis of the observations it makes.

There is only a problem if we have prefixes $\bar{z}_1$ and $\bar{z}_2$ of traces from $M(\bar{x})$ such that the actions of the tester at p should differ after $\bar{z}_1$ and $\bar{z}_2$ but this tester cannot distinguish between them ($\pi_p(\bar{z}_1) = \pi_p(\bar{z}_2)$).

Definition 3 *Given FSM M an input sequence $\bar{x}$ is* controllable *for M if there do not exist $\bar{z}_1, \bar{z}_2 \in pre(M(\bar{x}))$ such that $|\bar{z}_1| \neq |\bar{z}_2|$ and the next input to be applied after $\bar{z}_1$ is to be applied at a port p such that $\pi_p(\bar{z}_1) = \pi_p(\bar{z}_2)$. Where M is clear from the context we say that $\bar{x}$ is controllable.*

This is similar to the definition in [13] for a restricted form of IOTS in which a transition can only send output to one port[2]. It says that there cannot be traces $\bar{z}_1$ and $\bar{z}_2$ of different length, and so are to be followed by different inputs in $\bar{x}$, which look identical to the tester that should provide the next input after $\bar{z}_1$. As shown above, in M_1 sequence $x^U x^U x^L$ is not controllable. In contrast, if we consider $x^L x^U x^L x^U$ we find that the traces are $x^L/(y^U,y^L)x^U/(-,y^L)x^L/(y^U,-)x^U/(-,y^L)$ and $x^L/(y^U,y^L)x^U/(-,y^L)x^L/(y^U,y^L)x^U/(-,y^L)$ and so $x^L x^U x^L x^U$ is controllable.

[2]In general we cannot use such IOTSs to represent FSMs since if we order the outputs in a tuple from Y we can introduce what appear to be new controllability problems.

Proposition 1 *If $\bar{x}$ is controllable for FSM M then every trace in $M(\bar{x})$ is controllable.*

Proof: Proof by contradiction: assume that $\bar{x}$ is controllable for M and there exists a trace in $M(\bar{x})$ that has prefix $\bar{z}x_i/y_i x_{i+1}/y_{i+1}$ such that $port(x_{i+1}) \notin ports(x_i/y_i)$. But we can set $\bar{z}_1 = \bar{z}x_i/y_i$ and $\bar{z}_2 = \bar{z}$, $|\bar{z}_1| \neq |\bar{z}_2|$, and $\bar{z}_1$ should be followed by input at port $p = port(x_{i+1})$ such that $\pi_p(\bar{z}_1) = \pi_p(\bar{z}_2)$, providing a contradiction. ■

The following result will be useful.

Proposition 2 *If there exists $\bar{z}_1, \bar{z}_2 \in pre(M(\bar{x}))$ such that $|\bar{z}_1| \neq |\bar{z}_2|$ and the next input x_i in $\bar{x}$ to be applied after $\bar{z}_1$ is to be applied at a port p such that $\pi_p(\bar{z}_1) = \pi_p(\bar{z}_2)$ then $|\bar{z}_1| > |\bar{z}_2|$.*

Proof: This follows from observing that if $|\bar{z}_2| > |\bar{z}_1|$ then the inputs at p in $x_1, \ldots, x_i$ are in $\pi_p(\bar{z}_2)$ but x_i is not in $\pi_p(\bar{z}_1)$, giving a contradiction. ■

We can now prove that an input sequence being controllable is a necessary and sufficient condition for each tester knowing when to apply input.

Definition 4 *Given FSM M and $\bar{z} = x_1/y_1, \ldots, x_k/y_k \in L(M)$, for $1 < i \leq k$ the tester at $p = port(x_i)$ can* determine when to apply *$x_i \in X_p$ based on the observation of $\pi_p(x_1/y_1, \ldots, x_{i-1}/y_{i-1})$ if every trace in $pre(M(x_1, \ldots, x_k))$ in which the tester observes $\pi_p(x_1/y_1, \ldots, x_{i-1}/y_{i-1})$ has input portion $x_1, \ldots, x_{i-1}$.*

Proposition 3 *Given FSM M and input sequence $\bar{x} = x_1 \ldots x_k$, $\bar{x}$ is controllable for M if and only if for every $x_1/y_1 \ldots x_k/y_k \in M(\bar{x})$ and $1 < i \leq k$ the tester at $p = port(x_i)$ can determine when to apply x_i based on the observation of $\pi_p(x_1/y_1 \ldots x_{i-1}/y_{i-1})$.*

Proof: First we assume that $\bar{x}$ is controllable for M. By Definition 3, the tester at $port(x_i)$ knows when to apply x_i.

Now assume that $\bar{x}$ is not controllable for M. Thus there exist $\bar{z}_1, \bar{z}_2 \in pre(M(\bar{x}))$ such that $|\bar{z}_1| \neq |\bar{z}_2|$ and the next input to be applied after $\bar{z}_1$ is to be applied at a port p such that $\pi_p(\bar{z}_1) = \pi_p(\bar{z}_2)$. By Proposition 2 we know that $|\bar{z}_1| > |\bar{z}_2|$. The tester at port p does not know when to apply the input that follows $\bar{z}_1$, since it cannot differentiate between traces $\bar{z}_1$ and $\bar{z}_2$. The result thus follows. ■

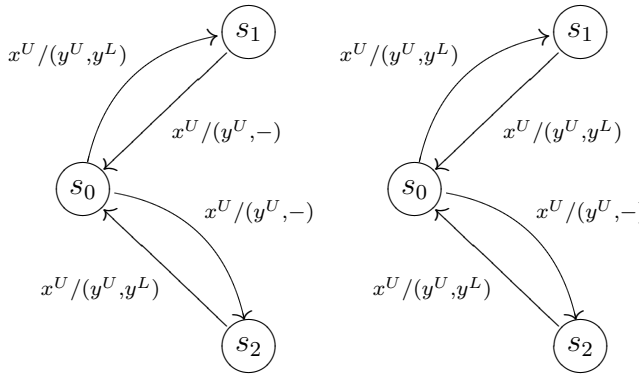

Fig. 2. The FSMs N_2 and M_2

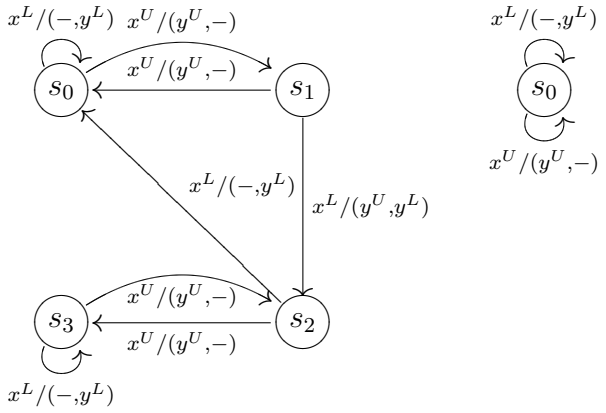

Fig. 3. The FSMs M_3 and M_4

When testing from a DFSM, if a controllable input sequence $\bar{x}$ leads to different traces from two states or DFSMs then there is a prefix of $\bar{x}$ that leads to a failure being observed [10]. This means that using prefixes of $\bar{x}$ allows us to overcome the fault masking that can be introduced: if a controllable input sequence $\bar{x}$ leads to a trace in the SUT N that is not in the DFSM specification M then there is a prefix of $\bar{x}$ that leads to a trace of N that is not equivalent to any trace of M under $\sim$. This result does not hold for FSMs.

Proposition 4 *Let us suppose that when controllable input sequence $\bar{x}$ is applied to the SUT it can produce a trace that is not in $M(\bar{x})$. It is possible that for every prefix $\bar{x}'$ of $\bar{x}$, when $\bar{x}'$ is applied to the SUT it must produce a trace $\bar{z}'$ such that there exists $\bar{z} \in M(\bar{x}')$ with $\bar{z} \sim \bar{z}'$.*

Proof: Consider the FSMs N_2 and M_2 shown in Figure 2 in which N_2 is the SUT and M_2 the model of the required behaviour. Then $L(N_2) \setminus L(M_2)$ contains $x^U/(y^U,y^L)x^U/(y^U,-)$. First, every proper prefix of this trace is a trace of M_2. The result follows from observing that in the distributed test architecture we cannot differentiate between this trace and the trace $x^U/(y^U,-)x^U/(y^U,y^L)$ in $L(M_2)$ since they are equivalent under $\sim$. ■

The use of the distributed test architecture affects the ability of testing to distinguish an FSM SUT and an FSM specification even if there cannot be controllability problems.

Proposition 5 *Let M and N be completely-specified FSMs with the same input and output alphabets. If N is a reduction of FSM M then N is a local reduction of M. However, N can be a local reduction of FSM M and not a reduction of M.*

Proof: The first part is immediate from the definitions of N being a reduction of M and N being a local reduction of M and the second part follows from Proposition 4. ■

V. Controllable input sequences

In this section we show how one can decide whether an input sequence is controllable, give a test generation algorithm and discuss how one can produce testers for a controllable input sequence.

A. Deciding whether a sequence is controllable

For input sequence $\bar{x} = x_1, \ldots, x_k$ and $1 \leq i < k$ it is sufficient to check each $\bar{z}_1 = x_1/y_1 \ldots x_i/y_i \in M(\bar{x})$ to see whether there is a shorter prefix $\bar{z}_2$ of a trace in $M(\bar{x})$ such that $\pi_p(\bar{z}_1) = \pi_p(\bar{z}_2)$ for port $p = port(x_{i+1})$ [13]. However, generating $M(\bar{x})$ can lead to a combinatorial explosion and so here we give an algorithm that does not require us to construct this set. We achieve this by, for an input sequence $\bar{x}$ and FSM M, constructing a (partially-specified) FSM $M_{\bar{x}}$ that represents all possible responses of M to $\bar{x}$. Algorithm 1 achieves this through k iterations. The ith iteration takes the set S_{i-1} of states reached by $x_1, \ldots, x_{i-1}$ and determines which states of M are reachable from these by x_i, forming the set S_i. Transitions are added between states in S_{i-1} and S_i.

The following is clear from the way $M_{\bar{x}}$ is constructed.

Algorithm 1

1) *Input* $\bar{x} = x_1 \ldots x_k$, M
2) *Output FSM* $M_{\bar{x}} = (S_{\bar{x}}, s_{00}, X, Y, T_{\bar{x}})$.
3) *Let* $S_{\bar{x}} = \{s_{00}\}$; $T_{\bar{x}} = \emptyset$; $S_0 = \{s_{00}\}$. Comment: We represent state s_j reached by $x_1, \ldots, x_i$ as state s_{ij} in S_i.
4) *For* $i = 1$ *to* k
   a) *Let* $S_i = \emptyset$
   b) *For all* $s_{i-1j} \in S_{i-1}$ *and transition* $(s_j, s_{j'}, x_i/y) \in T$ *let* $S_i = S_i \cup \{s_{ij'}\}$ *and* $T_{\bar{x}} = T_{\bar{x}} \cup (s_{i-1j}, s_{ij'}, x_i/y)$
   c) *Endfor*
   d) *Let* $S_{\bar{x}} = S_{\bar{x}} \cup S_i$
5) *Endfor*
6) *Return* $M_{\bar{x}} = (S_{\bar{x}}, s_{00}, X, Y, T_{\bar{x}})$

Fig. 4. Generating $M_{\bar{x}}$

Proposition 6 *Let us suppose that we apply Algorithm 1 to FSM* M *and input sequence* $\bar{x} = x_1, \ldots . x_k$ *and* $M_{\bar{x}}$ *is returned. Then* $\bar{x}'/\bar{y}' \in \mathcal{L}(M_{\bar{x}})$ *reaches a state in* S_i *if and only if* $\bar{x}' = x_1, \ldots, x_i$ *and* $\bar{x}'/\bar{y}' \in \mathcal{L}(M)$.

Proposition 7 *Given an FSM with transition set* T *and input sequence with length* k*, Algorithm 1 has time complexity of* $O(k|T|)$.

Proof: There are k iterations of the outer loop and each iteration contains at most one step for each transition in T. The result thus follows. ■

Given $p \in \mathcal{P}$ and $1 \leq i \leq k$ we define a finite automaton whose language corresponds to the observations that can be made at p if the prefix of $\bar{x}$ of length i is applied to M. However, first we define finite automata. A *finite automaton (FA)* N is defined by a tuple (S, s_0, A, T, F) in which S is a finite set of states; $s_0 \in S$ is the initial state; A is the finite alphabet; T is the set of transitions, each transition being of the form (s, s', a) for $s, s' \in S$ and $a \in A \cup \{-\}$; and F is the set of final states. An FSM is a type of FA and so we will use corresponding terminology such as a path and the label of a path. The FA N defines the regular language $\mathcal{L}(N)$ of labels of paths that have starting state s_0 and whose ending state is in F; instances of $-$ are not included in the label. We now explain how the FA that we will use can be constructed.

Let us suppose that we have FSM M, input sequence $\bar{x} = x_1, \ldots, x_k$, the FSM $M_{\bar{x}} = (S_{\bar{x}}, s_{00}, X, Y, T_{\bar{x}})$ returned by Algorithm 1, $1 \leq i \leq k$, and port $p \in \mathcal{P}$. We will define FA $N(M, \bar{x}, i, p) = (S_{\bar{x}i}, s_{00}, X_p \cup Y_p, T_{\bar{x}i}, S_i)$ in which $\mathcal{L}(N(M, \bar{x}, i, p))$ is the projection onto p of all possible response of M to $x_1, \ldots, x_i$ and so $\mathcal{L}(N(M, \bar{x}, i, p)) = \pi_p(M(x_1, \ldots, x_i))$. We can construct $N(M, \bar{x}, i, p)$ when building $M_{\bar{x}}$ by:

1) Returning $N(M, \bar{x}, i, p)$ when the iteration for x_i has finished; and
2) For each transition $(s, s', x/y)$ added to $T_{\bar{x}}$ before the iteration for x_i has finished add transitions to $N(M, \bar{x}, i, p)$ from s to s' that form a path with label $\pi_p(x/y)$. There are three cases here: if $\pi_p(x/y) = \epsilon$ then add a single transition with label $-$; if $\pi_p(x/y) = a$ for some $a \in X_p \cup Y_p$ then add a single transition from s to s' with label a; if $\pi_p(x/y) = x_p y_p$ for some $x_p \in X_p$, $y_p \in Y_p$ then add a transition with label x_p from s to a new intermediate state s'' and a transition with label y_p from s'' to s'.

We now define FA $N'(M, \bar{x}, i, p)$ whose language is the projection onto p of possible responses of M to proper prefixes of $x_1, \ldots, x_i$. To construct $N'(M, \bar{x}, i, p)$ it is sufficient to take $N(M, \bar{x}, i, p)$ and make $S_0 \cup \ldots \cup S_{i-1}$ the set of final states. We can now give a condition that allows us to decide whether $\bar{x}$ causes a controllability problem using these FA.

Proposition 8 *The input sequence* $\bar{x} = x_1, \ldots, x_k$ *causes a controllability problem with* M *if and only if there exist* $1 < i \leq k$ *and port* p *such that* $x_i \in X_p$ *and* $\mathcal{L}(N(M, \bar{x}, i-1, p)) \cap \mathcal{L}(N'(M, \bar{x}, i-1, p)) \neq \emptyset$. *In addition, it is possible to decide this in time that is polynomial in terms of* k *and the number of transitions of* M.

Proof: By Proposition 2, input x_i at p causes a controllability problem if and only if there exist $\bar{z}_1 \in M(x_1, \ldots, x_{i-1})$, $j < i$, and $\bar{z}_2 \in M(x_1, \ldots, x_{j-1})$ such that $\pi_p(\bar{z}_1) = \pi_p(\bar{z}_2)$. But this holds if and only if there exists $j < i$ such that $\pi_p(M(x_1, \ldots, x_{i-1})) \cap \pi_p(M(x_1, \ldots, x_{j-1})) \neq \emptyset$. The first part of the result follows from observing that $\pi_p(M(x_1, \ldots, x_{i-1})) = \mathcal{L}(N(M, \bar{x}, i-1, p))$ and $\cup_{j<i} \pi_p(M(x_1, \ldots, x_{j-1})) = \mathcal{L}(N'(M, \bar{x}, i-1, p))$.

For the complexity result first observe that the FA $N(M, \bar{x}, i-1, p)$ and $N'(M, \bar{x}, i-1, p)$ can be

Algorithm 2

1) *Input* M, $\bar{x} = x_1 \dots x_k$
2) *Output whether* $\bar{x}$ *is controllable for* M
3) *Result = True*
4) *For* $i = 2$ *to* k
   a) *Let* $p = port(x_i)$
   b) *If* $\mathcal{L}(N(M, \bar{x}, i-1, p)) \cap \mathcal{L}(N'(M, \bar{x}, i-1, p)) \neq \emptyset$ *then Result = False*
5) *Endfor*
6) *Return Result*

Fig. 5. Deciding whether $\bar{x}$ is controllable

constructed alongside Algorithm 1. Second in order to decide whether $\mathcal{L}(F_1) \cap \mathcal{L}(F_2) = \emptyset$ for FA F_1 and F_2 with n_1 and n_2 states respectively it is sufficient to use a product automaton with at most $n_1 n_2$ states and then decide whether the corresponding language is empty by determining whether any of its final states are reachable (see, for example, [16]). ■

The algorithm is summarised in Algorithm 2.

B. Test generation

Algorithm 3 returns a controllable input sequence. The algorithm iterates; at the beginning of each iteration it can stop and otherwise we check to see which tester can be the source of the next input. If the previous input was at p then we use Algorithm 2 to determine whether extending with input at $q \neq p$ would lead to a controllability problem[3]. We find a set P of ports to which we can send an input without causing a controllability problem and choose an input from some X_q for $q \in P$.

Now consider the application of Algorithm 3 to M_1 and assume we initially choose input x^U. On the next iteration we find that input at L after x^U would lead to a controllability problem so if we are to continue we must use x^U, leading to $x^U x^U$. Again, input at L will cause a controllability problem and so we could choose input x^U, leading to $x^U x^U x^U$. We could now terminate, returning this input sequence. The algorithm contains choices and importantly, if $\bar{x}$ is a controllable input sequence then $\bar{x}$ can be returned by Algorithm 3.

Proposition 9 *Given FSM M, an input sequence $\bar{x}$ is controllable if and only if it can be returned by Algorithm 3.*

[3]Whether an input in X_q causes a controllability problem after $x_1, \dots, x_i$ does not depend on the actual input used.

Algorithm 3

1) *Input FSM* M.
2) *Either return* ϵ *and terminate or choose an initial input* x *and set* $\bar{x} = x$.
3) *While (true)*
4) *Return* $\bar{x}$ *and terminate or do the following:*
5) *Let* x' *denote the last input in* $\bar{x}$ *and let* $p = port(x')$.
6) $P = \{p\}$
7) *For all* $q \in \mathcal{P} \setminus \{p\}$ *do:*
   a) *If there do not exist* $\bar{z}_1, \bar{z}_2 \in M(\bar{x})$ *and a proper prefix* $\bar{z}_2'$ *of* $\bar{z}_2$ *such that* $\pi_q(\bar{z}_1) = \pi_q(\bar{z}_2')$ *then* $P = P \cup \{q\}$. Comment: We can adapt Algorithm 2 to decide this
8) *Choose* $q \in P$, *some* $x \in X_q$ *and set* $\bar{x} = \bar{x}x$.
9) *Endwhile*

Fig. 6. Generating controllable input sequences

Proof: This follows from noting that the algorithm ensures that when an input sequence is extended the new input sequence satisfies the condition for a sequence to be controllable but places no additional restrictions. ■

Test generation could be random or we might aim to satisfy a test objective. It may be possible to adapt techniques for testing from a nondeterministic finite state machine [17] or use game theory to guide the choice of next input (see, for example, [18]).

C. Generating the testers

Given controllable input sequence $\bar{x}$, we need to produce testers to place at the ports in order to apply $\bar{x}$. In contrast to the case with DFSMs, we may require the tester to be placed at port p to be adaptive. To see this, consider an FSM M and input sequence $x^U x^U x^L$ such that $M(x^U x^U x^L) = \{x^U/(y^U, y_1^L)x^U(-, y_1^L)x^L/(y^U, y^L),$
$x^U/(y^U, -)x^U/(-, y_2^L)x^L/(y^U, y^L)\}$. The tester at L sends input x^L after observing $y_1^L y_1^L$ or y_2^L.

For M and controllable $\bar{x} = x_1, \dots, x_k$, we can produce a tester for $p \in \mathcal{P}$ by using the FA $N(M, \bar{x}, k, p)$ that accepts $\pi_p(M(\bar{x}))$. Let us suppose, for example, that we want to produce testers for $\bar{x} = x^U x^U x^U$ when testing from M_1. $N(M_1, \bar{x}, 2, U)$ and $N(M_1, \bar{x}, 2, L)$ both have two

paths to a final state. In $N(M_1, \bar{x}, 2, U)$, the two paths in the FA that define the tester at U have labels $x^U y^U x^U x^U y^U$ and $x^U x^U x^U y^U$; for $N(M_1, \bar{x}, 2, L)$ the paths have labels $y^L y^L y^L$ and $y^L y^L$.

While the problem of generating testers has been solved for IOTSs [13], the previous approach would require all elements of $M(\bar{x})$ to be produced and this can lead to a combinatorial explosion.

VI. Conclusions

In the distributed test architecture the tester at port p only observes events at p, the testers cannot communicate with one another during testing and there is no global clock. This can introduce controllability problems, which have been studied for testing from deterministic finite state machines (DFSMs) and this paper has investigated them for nondeterministic finite state machines (FSMs).

As with DFSMs, a tester can only know when to apply an input if it was involved in the previous transition. However, when testing from FSM M with input sequence $\bar{x}$ an additional issue arises if the next required behaviours of the tester at p differ after two possible responses of M to prefixes of $\bar{x}$ and the tester cannot differentiate between these cases: it does not know which behaviour to use.

We investigated controllability problems and gave a polynomial time algorithm to decide whether an input sequence $\bar{x}$ is controllable for FSM M. We then gave a test generation algorithm that returns controllable input sequences. In order to apply input sequence $\bar{x}$ it is necessary to produce one tester for each port and we gave a polynomial time algorithm to do this. Previous algorithms for these problems, for testing from an IOTS [13], require us to produce all responses of M to $\bar{x}$ and can take time and space that is exponential in the length of $\bar{x}$.

There are several avenues for future work. When trying to achieving a test objective it may be possible to adapt techniques for testing from an FSM or by using game theory. Recent work has considered models in which operations can be triggered by inputs received from several ports [19] and it would be interesting to extend the results to such models. Finally, it is likely that insights will be gained by using the results in large industrial case studies.

References

[1] A. Y. Duale and M. U. Uyar, "A method enabling feasible conformance test sequence generation for EFSM models," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 614–627, 2004.

[2] W. Grieskamp, "Multi-paradigmatic model-based testing," in *Formal Approaches to Software Testing and Runtime Verification (FATES/RV 2006)*, ser. Lecture Notes in Computer Science, vol. 4262. Springer, 2006, pp. 1–19.

[3] D. Lee and M. Yannakakis, "Principles and methods of testing finite-state machines - a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1089–1123, 1996.

[4] J. Chen, R. M. Hierons, and H. Ural, "Overcoming observability problems in distributed test architectures," *Information Processing Letters*, vol. 98, pp. 177–182, 2006.

[5] R. Dssouli and G. von Bochmann, "Error detection with multiple observers," in *Protocol Specification, Testing and Verification V*. Elsevier Science (North Holland), 1985, pp. 483–494.

[6] R. M. Hierons and H. Ural, "Synchronized checking sequences based on UIO sequences," *Information and Software Technology*, vol. 45, no. 12, pp. 793–803, 2003.

[7] O. Rafiq and L. Cacciari, "Coordination algorithm for distributed testing," *The Journal of Supercomputing*, vol. 24, no. 2, pp. 203–211, 2003.

[8] H. Ural and C. Williams, "Constructing checking sequences for distributed testing," *Formal Aspects of Computing*, vol. 18, no. 1, pp. 84–101, 2006.

[9] 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*, 1998, pp. 80–85.

[10] R. M. Hierons and H. Ural, "The effect of the distributed test architecture on the power of testing," *The Computer Journal*, vol. 51, no. 4, pp. 497–510, 2008.

[11] J. L. Jacob, "Refinement of shared systems," in *The Theory and Practice of Refinement: Approaches to the Formal Development of Large-Scale Software Systems*, J. McDermid, Ed. Butterworths, 1989, pp. 27–36.

[12] M. A. Fecko, M. U. Uyar, A. S. Sethi, and P. D. Amer, "Conformance testing in systems with semicontrollable interfaces," *Annals of Telecommunications*, vol. 55, no. 2, pp. 70–83, 2000.

[13] R. M. Hierons, M. G. Merayo, and M. Núñez, "Controllable test cases for the distributed test architecture," in *6th International Symposium on Automated Technology for Verification and Analysis (ATVA 2008)*, ser. Lecture Notes in Computer Science. Springer, 2008, pp. 201–215.

[14] J. T. C. 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] R. M. Hierons, M. G. Merayo, and M. Núñez, "Implementation relations for the distributed test architecture," in *(TestCom/FATES 2008)*, ser. Lecture Notes in Computer Science, vol. 5047. Springer, 2008, pp. 200–215.

[16] M. Holzer and M. Kutrib, "State complexity of basic operations on nondeterministic finite automata," in *7th International Conference on the Implementation and Application of Automata (CIAA)*, ser. Lecture Notes in Computer Science, vol. 2608. Springer, 2002, pp. 148–157.

[17] A. Petrenko and N. Yevtushenko, "Testing from partial deterministic FSM specifications," *IEEE Transactions on Computers*, vol. 54, no. 9, pp. 1154–1165, 2005.

[18] R. Alur, C. Courcoubetis, and M. Yannakakis, "Distinguishing tests for nondeterministic and probabilistic machines," in *27th ACM Symposium on Theory of Computing*, 1995, pp. 363–372.

[19] S. Haar, C. Jard, and G.-V. Jourdan, "Testing input/output partial order automata," in *(TestCom/FATES 2007)*, ser. Lecture Notes in Computer Science, vol. 4581. Springer, 2007, pp. 171–185.